Abstract
In our previous work, the performance enhancement of enlarged FFT kernels was demonstrated with 48-256 point kernels on an Intel x86. The same approach is extended to SIMD compatible FFT kernels on an Intel AVX SIMD unit. SIMD-compatible enlarged FFT kernels (24, 48 and 72 points) are up to 30% faster than normal-sized kernels, and the behavior of performance enhancement is more irregular than non-SIMD cases.

Keywords: FFT, large FFT kernels, optimization, SIMD, source code generation

要旨
前論文で、従来よりも大きな48から256点のFFTカーネルの性能評価を行ったところ、Intel64 CPUにて予想を超える性能向上を見た。今回、現代の高速プロセッサの活用に於いて必須であるSIMDを利用する24、48、72点のカーネルをIntel CPUのAVX SIMDユニットにて評価したところ最大30%の性能改善が見られた。ただし、性能向上率はより不規則な挙動をとるようになった。

キーワード：FFT、大型FFTカーネル、最適化、SIMD、ソースコード生成
1. Introduction

The performance of Fast Fourier Transform (FFT) is one of the key issues in supercomputing[1]. However, it is now quite difficult to further improve the FFT algorithms or implementation of them. We have tested a simpler approach, that is, an optimization of the kernel size, which is an adjustable parameter of the FFT algorithm, and successfully derived the enhancement of performance[2] in an Intel chip. FFT is a recursive algorithm, where a $k$-point FFT is executed by decomposing it into numerous $j$-point FFTs ($k > j$), until $j$ is small enough for direct calculation. In this recursion, a $j$-point FFT kernel is a level of recursion that is realized by a self-calling subroutine in which FFT is “hardcoded”. In the most basic Cooley-Tukey FFT, $j$ is 2.

The value(s) of $j$ is referred to as “kernel-size” of FFT that can be enlarged by means of, for example, fusing two stages of recursion to one. The size of the kernel is known to affect the FFT performance through several mechanisms[2], and, in general, large kernels are believed to be faster in ideal conditions[3][4]. However, benchmarks of kernels greater than $k=48$ seem to be rarely reported, due to, perhaps, FFT kernels larger than the capacity of the registers of CPU did not appear promising. One of the most advanced FFT suites available, FFTW[5], can handle many different combinations of kernel sizes for a given FFT size: FFTW is capable of generating kernels with general sizes and adjusting the size and combination of kernels such that the total performance of FFTW is (automatically) maximized. In our previous report[2], we used FFTW to show an unexpected effectiveness of enlarged kernels with $j=48$–256 for Intel Corei7 processors which has only 16 general registers. To extend this result and draw a generally applicable conclusion, it is suitable to do similar experiments on various computers with (different architectures and) different numbers of registers in CPU. By the way, for the case of general (non-FFT) processing, these reports[6][7] have already discussed the relationship between the register number and the performance based on experimental data.

However, there are many difficulties to carry out this type of experiment. Computers with less than 16 registers, (e.g. Motorola MC68000 series), are obsolete and not compatible with many software tools required by FFTW. Computers with more than 16 registers are available but pose an entirely different and more severe problem. In latest CPU cores, maximum register usage is somehow related to full exploitation of SIMD (single instruction multiple data) computing, that is typically shown in Fujitsu’s FX10, FX100, and K computer[8]. Although “SIMD” originally meant a type of parallel computing in Flynn's taxonomy[9], now it refers to the additional unit in CPU core that executes single numerical operations for multiple operands in a parallel manner. Regarding the number of the registers, there are similar situations for microprocessors from other vendors with their own SIMD unit. For example, the Intel AVX (Advanced Vector eXtension) SIMD unit[10] has additional 64 double-precision registers. More importantly, in these computers, SIMD is also a prerequisite for a greater IPC (instruction per cycle) and maximal numerical performance because the theoretical performance of a SIMD unit is typically 2–4 times larger than that of the main part of the CPU.
Considering these, we have generated and tested \( j = 24, 48, \) and 72 enlarged and SIMD-compatible kernels on FFTW on an Intel Chip. Of course, this can be at least an indirect test of enlarged FFT kernels under large number of registers. The result is relevant for any FFT usage on various supercomputers that are based on SIMD-enhanced microprocessors designed and manufactured by Intel, AMD, IBM and Fujitsu. Future supercomputing based on, for example, ARM NEON[11] SIMD architectures is also relevant.

2. Method

In a SIMD-compatible fashion, \( j = 24, 48, \) and 72 enlarged FFT kernels are prepared as an “add-on” to a state-of-the-art FFT code suite, FFTW (v3.3.3), that is compiled on an Intel64 computer. FFTW automatically selects the best combination of kernels to realize given size of FFT: therefore, these additional kernels are expected to be selected when \( k \) (the size of whole FFT) is a multiple of \( j \) as one level of the recursions. It was impossible to find a better experimental design (with, for example, manual derivations of FFT kernels and direct control of the kernel size). The details of kernel preparation and performance evaluation are explained in detail as follows.

Although not widely recognized, FFTW is not just a FFT subroutine library but contains a complete code-generator for FFT kernels, called “fftgen”, implemented in the OCaml language[12] (actually, ocamlopt, the OCaml compiler is necessary): one can use fftgen to automatically generate FFT kernels with ANY size. First, in a preliminary experiment, we have confirmed that in its “-simd” mode, fftgen can generate FFT kernels that fully utilize SIMD units of Intel AVX. Since no source-code of greater than \( j = 8 \) FFT kernel is found in public domain in SIMD-compatible forms and the kernel generated by FFTW is strongly dependent on other elements in FFTW suite, all experiments had to be fully based on FFTW.

We generated and tested \( j = 24, 48, \) and 72 point FFT kernels in an independent manner. That is, for each of these three sizes, a FFTW binary with single additional SIMD-compatible kernel is compiled and tested for \( k = 144 \)~2036 1D FFT with interval of 24. Since none of these enlarged kernel is found in the default setting of the SIMD-compatible part of FFTW, any positive performance change by these kernels should be detected. The actual generation of additional SIMD-compatible FFT kernels is carried out by modifying the SIMD-compatible kernel list (dft/simd/codlist.mk) and executing “sh bootstrap.sh” in FFTW suite top-level directory, before compilation of FFTW. The process of actual kernel generation (schematically shown in Fig.1) is fully automated and no further human intervention is required.
Here, it is worthwhile to discuss the reason why these SIMD-compatible kernels have to be specially generated aside from the FFT kernels we have already generated in previous work. In short, it was impossible for the compilers to convert normal FFT kernels to SIMD compatible executable-codes, because FFT kernels generated by fftgen usually lack loops. The detailed discussion is as follows.

Many compilers need the parallelism that are explicitly designated as loops (and sometimes the parallel directives) for their auto-vectorization feature to utilize the SIMD units. That is, from the programmers viewpoint, the function and usage of SIMD units are in many respect reminiscent of the vector supercomputers in 1980~ 90s[13], although the vector operands may not be processed in a pipelined manner in modern SIMD units. For example, gcc has been with a severe limitation in exploiting Intel SIMD functions for many generations, but the latest versions of gcc (4.4 or later) are capable of loop auto-vectorization. In large FFT kernels, many FMA (fused-multiply-and) operations are executed on different operands, but the compilers cannot exploit this type of parallelism because they are not described in loops. The compilers of Fujitsu supercomputers also lack the ability to find parallelism for SIMD from a sequence of operations part of which might be independent. This problem is resolved when fftgen is executed in “-simd” mode by means of introducing an “outer” loops, that is provided as a “SIMD intrinsic” commands, to FFT kernels. In “-simd” mode, fftgen seems to be generating codes that execute a number of independent j-point FFT kernels in a SIMD unit simultaneously. Therefore, the net performance is improved only when FFT requires enough number, i.e. more than the “width” of SIMD hardware, of j-point FFT kernels. Considering this, in this report, a j-point SIMD-compatible kernel is tested mainly when k>4j regime, because an acceleration is theoretically possible only in this regime.
The generation of FFTW kernels with modified kernel sizes is performed in a Linux PC (Ubuntu14.04, based on Corei7-3770S at 3.1GHz). The performance test is performed by “bench” program included in the FFTW suite in a different computer (2.5GHz Xeon) due to a technical reason. The generated kernels are compiled by the gnu C compiler (gcc V4, on linuxPC). No multithread programming is involved to isolate the effect of these element. Standard optimization options are used (-03 for gcc). For a single condition, 40 independent FFT executions are performed, and according to the execution time we calculate the relative performance enhancement of enlarged FFT kernels compared to normal FFT kernels, both under SIMD environment.

3. Results
The plots in Fig.2 (A) demonstrates the relative performance index (i.e. ratio of the execution times to complete FFT for given $k$) of the FFTW enhanced by enlarged kernels compared to FFTW compiled with the default settings for SIMD computation, observed in an Intel Xeon processor. Enlarged kernels improve the performance significantly: It should be also noted that in general, SIMD-enabled FFTW with normal kernels, our baseline in this test, is already about twice faster than FFTW without SIMD.

Large kernels show a strong performance increase for many $k>4j$ conditions. As easily expected, $k<4j$ conditions show very little improvement as $j=72$ most clearly demonstrates. The degree of performance increase may be a strongly decreasing function of the size of added kernels, although $j=256$ kernel cannot be tested because it is already in FFTW. Many performance peaks appear on $k$ is a multiple of $j$, proving that this method is as inflexible as non-SIMD cases. In summary, this data clearly shows that FFT can be improved by simply using larger kernels even when SIMD in a microprocessor is activated.

However, it seems to be difficult to extract a generally applicable rule to predict the performance peaks. Considering that the hardware SIMD-width of AVX is 4, the X-axis of Fig.2 (A) is “folded” such that $k'=(k/j) \mod 4$ is the X-axis of Fig.2 (B). Strong performance enhancement is observed almost always at $k'=0$, where $k$ is a multiple of 4*(kernel size). However, size 24 kernel is not fully following this rule, and some of $k'=2$ cases are also accelerated for kernel size 48 and 72. The reason of these complex behavior is unknown, but should be related to the data transfer process between SIMD- and normal registers. It should be also noted that performance enhancement does not always happens even if $k'=0$. We do not have measures to properly analyze these behavior in detail, as well as the detailed cause of performance drops observed for a number of cases.
Fig. 2  Relative performance of 1DFFT with enlarged SIMD kernels as (A) a function of FFT size and (B) a function of the number of the operands that incompletely filling the SIMD unit.
4. Discussion

For the effect of the number of the registers, the only conclusion available from this indirect test is that, in IA64, more registers that are available with SIMD is not harmful for the performance enhancement. Despite the severe degradation of flexibility, the simple approach of acceleration by means of enlarged FFT kernels is proven to be feasible in an Intel-AVX SIMD unit in an IA64 microprocessor. It seems there is no problem in generating larger than 72-point SIMD-compatible kernels for fftgen: the execution time of fftgen will be increased in a dramatic but not prohibitive way. However, the performance improvement is smaller in 72-point kernels than 48-point kernels, which strongly suggests that the upper-limit of effective kernel size is near.

Some time, enlarged FFT kernels are ineffective even when the “loop length” $k/j$ is large enough. The reason is unknown but may be related to other FFT kernels in FFTW that is larger than $j$ (that is, those cases are already SIMD-optimized very well in the default settings). Other possibility is the data transfer process between SIMD registers and normal registers that cannot be adjusted (although FFTW manual insists that the kernel generated by FFTW is always optimum in terms of such a data movement for wide variety of hardware designs).

5. Summary and Future Subjects

In theory, introduction of a SIMD-compatible kernel is the most effective when the kernel is executed for numerous independent operands. This condition occurs under (1) extremely large FFT (2) high-dimensional FFT and (3) numerous independent FFTs. The TOMBO[14] code realizes a situation similar to case (3) because it performs many 64- and 48-point 3D FFTs for different operands, that are already distributed among nodes in an MPI environment. In other words, the SIMD-compatible 48-point kernel can be easily introduced to TOMBO without disturbing its parallelization structure. Largely the same situation can be expected in many other parallelized ab-initio codes using FFTW, which will also be ideal test cases. In theory, executing one large FFT by multiple nodes by SIMD-compatible kernels is also possible. However, reduction of computing time in each node may complicate the optimization of internode communication. Monte Carlo simulations of spin systems with long-range interactions evaluated by FFT may also be accelerated.

It is quite imperative to extend this result to supercomputers based on processors with different SIMD units, such as Fujitsu’s FX10, FX100 and K computer. However, SIMD compatible kernel generation was attempted in vain for these computers due to the lack of “SIMD-intrinsics”. The situation is largely the same for Hitachi SR16000 (IBM Power7 architecture). An additional problem that should be considered here is that the methods for direct observation of the data movement to/from SIMD registers are not provided and the deep analysis of this issue is hindered by the complex structure of modern microprocessors. Possibly, the simpler structure of GPGPU can be a better tool to tackle this problem.
ACKNOWLEDGEMENTS
This research used computational resources of the HPCI system provided by Tohoku University Cyberscience Center through the HPCI System Research Project (Project ID: hp140061).

References