Vol.2012-ARC-200 No.18
Vol.2012-OS-121 No.18
2012/5/8

Deep DVS in FU Array by Covering Process Variations with Data-Path Auto-fix

Jun Yao[†1] and Yasuhiko Nakashima[†1]

Process variability in advanced semiconductor technologies has become a major constraint for improving the working frequency, especially under voltage scaling. In this research, based on a full understanding of unit utilization in an FU array accelerator, we propose a method to auto-fix the execution hardware data-path so as to include units with tolerable process variations. This method can help apply a deep DVS to achieve a nearly optimal energy reduction which is usually not possible due to the worst case process variation. A preliminary simulation result indicates that even after an aggressive power-gating, a 21.9% energy reduction without any performance impact can still be achieved by avoiding adverse and using beneficial process variations.

1. Introduction

Nowadays, it has been well acknowledged that performance scaling is becoming increasingly difficult under the challenging circumstance that the power consumption climbs to the limitation of heat dissipation and results in an intolerable short battery life. Accordingly, the trends of computer markets have gradually switched from a performance targeting goal to an energy efficiency targeting one, which puts emphases on both processing ability and power consumption.

Many methods have been provided to reduce the power consumption without sacrificing major working ability. One possible way is to bring down the supply voltage in the microprocessor by applying dynamic voltage scaling (DVS). The voltage contributes a quadric aspect in the dynamic power part of the electronic device, as calculated by $P = \alpha CV^2 f$[1)]. However, as the voltage scaling will also affect the working speed of devices, the most effective condition to apply voltage scaling is working under some known performance requirement with deadlines, such as video decoding. Other than the deadline consideration, another kind of voltage controls can be achieved by using program behavior analysis, such as bringing down supply voltage under relatively long low-level cache misses[2)], or under a long or short term program phase prediction[3)–5)]. Most of these method share a common consideration that general purpose processors give an over-design of working ability, and opportunities of voltage down-scaling can be found to achieve an optimization of the energy efficiency.

Another promising technology to reduce over-design point dynamically is via setup error detection logics, such as Razor Flip-Flops (FFs)[6)]. Razor-FF detects the setup timing error when DVS is applied in a pipeline processor. Therefore, the voltage can be aggressively set until the setup timing error rate reaches an intolerable level. It helps the processor remove all systematic margins, introduced by design precautions. The power consumption can be thus reduced by setting the voltage to reflect the processor's actual switching speed.

In this research, we try to move one step further in the process technology advancing road map. As technologies have advanced lower than 45nm level, other issues like process variability has become increasingly important. As has been analyzed in papers 7)–9), due to process variability, when working under a reduced voltage, the delays of the circuits tend to differ more than under the original supply voltage.

Accordingly, when semiconductor technologies keep advancing, the increasing process variability will add difficulties to reach an optimized voltage down-scaling. This is because the working frequency is determined by the worst-case unit, which is the pipeline stage with the longest critical path and affected by the worst adverse variation. Even when some circuits are affected by the beneficial process variations, their ability to switch fast is wasted by the slowing-down circuits. Simply applying Razor-FF can not solve the problem caused by process variability and straightforwardly, asynchronous circuits turn out to be a solution as they do not rely on a global synchronization.

Facing the possible inability of DVS in future processors with a large process variability, we propose to maintain the continuous DVS application by an architectural method to dynamically auto-fix the data-path that originally contains intolerable process variability units. Specifically, by treating the units with

†1 Nara Institute of Science and Technology

unfavored large process variation as permanent defected ones, it is possible to use architectural means to locate and remove them dynamically from the data-path. After this, only units with tolerable or even beneficial process variations are included in the data-path, which therefore achieves a process variation-free execution.

Accordingly, there are two main contributions in this work:

(1) The working voltage can be aggressively optimized to a level at which both design margins and adverse process variations are considered and removed.

(2) Usually, process variability is unfavored due to its uncertainty to affect the circuit reliability. However, by our method, we can even detect and use the beneficial process variations to achieve a deep DVS application.

According to the nature of this method, it requires a processor which periodically works on a fixed data-path. The best place to apply it is inside a CGRA architecture with a reconfigurable function unit (FU) array[10]. We give detailed tuning method to help an FU array processor to achieve optimized voltage setting and mapping. In addition, we also explore possibly applying this method with the fusion of many-core architectures.

The paper is organized as follows. Section 2 introduces the background and related techniques which are employed in this research. Section 3 gives the detailed architecture and algorithms to locate and remove intolerable adverse process variations. In addition, we try to extend the applicability of our method by indicating several possible extensions. Section 4 gives the preliminary energy saving results of the proposed method. Finally, Section 5 concludes the whole paper.

2. Background Techniques

2.1 Process Variability

Accordingly to papers 7)–9), the data-path delay shift can be categorized into systematic and random ones. Changing body-bias is a common way to re-shift the systematic variability, as neighbor circuits usually demonstrate a similar offset from this kind of variation. However, compared to the systematic one, the random variability is more unpredictable and tends to show no space correlation. Even side-to-side transistors may have different offsets. Therefore, this kind of random variability is hard for circuit level ways to re-shift and it can be expected to increase along the advances of process technology. In this work, we mainly focus on architectural ways to tolerate or even make good use of this kind of variability.

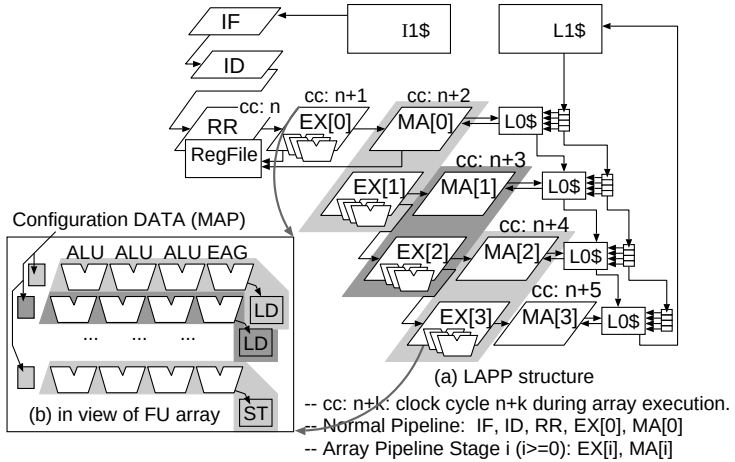

Fig. 1 Structure of LAPP.

2.2 LAPP: an FU Array Accelerator

To tune for a good voltage and frequency set, our method will require an understanding of the program that will be executed inside the processor in the next execution phase. To be more specific, we are trying to tune the program mapping on a large pool of processing elements to achieve a process variability free execution. Linear Array Pipeline Processor (LAPP)[11] is used as the baseline processor for this purpose. It is a specific accelerator implementation of a reconfigurable FU array based processor for a high-speed execution.

Fig. 1 shows the basic structure of LAPP. Basically, LAPP contains a normal VLIW pipeline, shown as IF, ID, RR, EX[0], and MA[0] in Fig. 1(a). Additionally, LAPP extends its EX and MA stages into extra working stages, as EX[i] and MA[i] ($i > 0$). The combination of these EXs and MAs take an FU array format, as depicted in Fig. 1(b). The FU array can be regarded as a series of array pipeline stages, in which each array pipeline stage represents EX[i] and MA[i] in Fig. 1(a).

By properly setting the configuration data, which is the mapping information

Vol.2012-ARC-200 No.18
Vol.2012-OS-121 No.18
2012/5/8

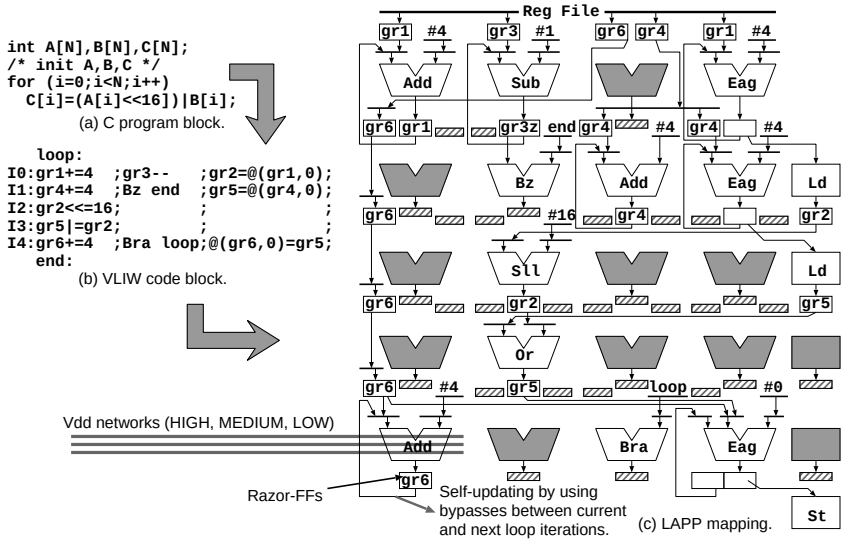

Fig. 2 Mapping a loop kernel onto LAPP.

in each array pipeline stage, it is possible to use LAPP to accelerate the hottest loop (**Fig. 2**(a)) inside the program. Specifically, the program block in Fig. 2(b) can be fit into the FU array, taking the mapping of Fig. 2(c).

As shown in Fig. 2(c), the loop kernel can be mapped onto 5 array pipeline stages. The FU array can overlap the executions of different loop iterations, following a pipeline concept. After the first 6 cycles, one loop iteration can be finished per cycle[⋆1], which indicates a significant speed-up by exploiting sufficient parallelism between iterations.

3. Proposal: Using Data-Path Auto-fix to Tolerate Process Variation

In this section, we introduce our proposed method which tries to detect and avoid using the units affected by the averse effect from process variability. Specifically, Razor-FF, which is originally used to detect setup timing variation to help eliminate system frequency design margins, is employed to find out the function units (FUs) that demonstrate adverse process variation. After that, we use architectural methods to collect these processed information and work out an auto-fix to optimally distribute the data graph onto units that are well constructed. In addition, when the voltage is aggressively down-scaled further, only units with beneficial process variation can survive the new setting. By detecting and using these units, it is possible to achieve a deep DVS application in the processor and gain a large amount of energy reduction.

The detailed method is introduced in following sections.

3.1 Models the Adverse Effect inside the Unit

As introduced in Section 2.1, we mainly focus on the random process variation which can not be easily eliminated by other techniques because the random effects are independent to the unit location. The basic unit that we look into is the FU inside the baseline architecture, as LAPP. To make it simple, we assume that the calculation FU include the following independent operation units:

(1) Logic & shift unit: AND, OR, XOR, SHIFT, and etc;
(2) Arithmetic unit: basic ones including ADD, SUB, and etc;
(3) multimedia processing unit: SADD, and etc;
(4) multiplication unit: multiplications which have relatively long path.

According to their different complexity in the hardware size, the four different units are expected to behave differently under a same random distribution of process variation. As from paper 12), a longer data path is relatively easier to conceal the process variability, because it may contain both beneficial and adverse effects from the process variability and make possibly an even when they are combined together. However, though the combined effect may possibly cancel each other, the possibility for worst case is still there. As the discussion of the possibility of process variation still remains as an open topic, in this research, we model it as follows:

(1) Assumption 1: The possibility of process variation directly connects to the longest delay of the unit. A long data path is likely to accumulate both adverse and beneficial effects. The process variation range can be calculated

⋆1 There are some restrictions such as no data dependence between different loop iterations except loop counters, and so on. Paper 11) has a clear definition of LAPP working situation.

Table 1 Process Variation and its Distribution .

	Delay (in FO4)	Normalized delay	Variation (100%V)
Logic&Shift	29.4	0.66	[-26%, -26%]
Arithmetic	39.2	0.88	[-35%, 35%]
Media	44.5	1.0	[-40%, 40%]
Multiplication	52.5	1.18	[-47%, 47%]

as $[-\Delta\tau, \Delta\tau]$, where $\Delta\tau = \frac{Delay}{Delay_{base}} \times \Delta\tau_{base}$.

(2) Assumption 2: While a long data path has shown a large range of process variations as from the above assumption, the possibilities for its worst and best case are however relatively lower than a short data path. Given that the data path contains n building blocks such as transistors with either variation $-\Delta\tau$ or $\Delta\tau$, the combined effect at variation $(k - n + k)\,\Delta\tau$ is $\binom{n}{k}$. Note that for simplicity, we assume that basic transistors always demonstrate the largest process variation. As $\sum_{0 \le k \le n} \binom{n}{k} = 2^n$, the possibility of each combined variation becomes $\binom{n}{k}/2^n$.

(3) Assumption 3: A lower supply voltage is supposed to amplify the process variation, which is caused by the shrinking of value $V_{DD} - V_{th}$, as stated in paper 8). Accordingly, we assume three voltages and their corresponding process variations, as ($V_{DD} = 100\%$, $Var = 40\%$), ($V_{DD} = 90\%$, $Var = 50\%$), and ($V_{DD} = 80\%$, $Var = 70\%$). Though the data are roughly assumed, they still follow the trend that when V_{DD} approaches near V_{th}, the variation becomes larger.

By the above assumptions, we get the process variations of the four operation units, as in **Table 1**. The delay data are taken by Synopsys Design Compiler, using a 180nm cell library. The data clearly show that the four kinds of operations have different delays. Note that the data path of each unit contains some shared operation code selection, input selection multiplexors, and latches to store data. For these additional logics, the difference between logic and multiplication is not as large as the one between a simple logic and multiplication.

3.2 Data-Path Auto-fix Method to Avoid Adverse Process Variations

The main idea of this research is to use architectural ways to detect and remove large adverse effects of process variability. From the other viewpoint, it is try to make a good use of beneficial process variations.

In this paper, voltage scaling technique is used to reduce power consumption. However, the working frequency is kept unchanged in order to maintain a same execution delay. It can be expected that setup timing will be largely violated under the low power execution. The reliability scheme is thus included to keep the execution correctness. The following sections introduce the method in detail.

3.2.1 Applying Multiple Supply Voltages in LAPP

In this research, we employ the architecture of LAPP to serve as our baseline processor which executes a mapped program inside a large bundle of FU resources. Specifically, in LAPP, a program inner loop is mapped onto its FU array to exploit maximum parallelism between loop iterations. Accordingly, LAPP will have a detailed understanding of its data-path and thus is able to tuning the data-path for the use of this paper. The mapping mechanism has been introduced in paper 11). As an example, Fig. 2 show a mapping result of a given program in detail. In this LAPP architecture, each array stage contains three execution FUs, one memory address generation (Eag) unit and a load/store unit. It requires 5 stages to hold this inner loop kernel inside.

In order to achieve energy efficiency, we add dynamic voltage control scheme in to the FU array of LAPP by introducing multiple supply voltages, similar to paper 13), 14). For simplicity, we assume that each array stage can select one of the three predetermined supply voltages as 100%, 90%, and 80% respectively. With a selection in Fig. 2, it is possible to connect the current array stage onto either of the three supply voltage networks. The reason that a stage level voltage domain is used as the voltage selection granularity is similar to paper 14), in which the authors indicate that a cell degree power gating control enlarges significantly the processor size while a coarse control loses power saving opportunities. The 90% and 80% supply voltages serve as low power execution modes. We also assume that without variation, the frequency shall be lowered to 80% and 50% of the original working frequency to reflect the degradation of circuit switching speed. However, in this research, we try to keep working under the original frequency even when the voltage is down-scaled. Therefore, the reliability scheme is required to address the setup timing errors, as introduced in the next sections.

3.2.2 Detecting Insufficient Frequency

The detection of setup timing violation is achieved by adding Razor-FF after each function unit, as shown in Fig. 2. When an FU can not meet the setup timing requirement, an error bit will be set and stored beside each array stage for latter usage. This test run can be carried out on each mapping by feeding the FUs with dummy data. One or several-cycle execution is already sufficient to set the error bits after Razor-FF.

Razor-FF uses a primary and shadow-latch structure to detect the possible setup error. Note that the error detection is done after the data reaches the primary latch in Razor-FF at the main clock edge. The comparison of two latches is performed in the next cycle. This part increases the FU array size accordingly while still leaving the critical path unchanged.

3.2.3 Addressing Insufficient Frequency by Auto-fixing Data-Path

In this research, we treat the detected timing error as a permanent error. After setting the error flags of the current instruction mapping by the first test run of the program, error resolution is triggered to auto-fix the data-path. This is the key point of this work to keep working under a lowered voltage without sacrificing working frequency and performance.

As shown in Fig. 2, we assume that in our baseline architecture LAPP, a VLIW instruction takes the form of one address generation, one branch operation and three calculations which can be either of basic arithmetic, logic, multiplication, and media processing. As address generation is usually not a full word-width calculation under a limited cache size in LAPP, we define that it has fewer chances to invoke a setup error than the other normal calculations. Similarly, branch operation gets jump target at the mapping phase and is thus not contributing to the determination of the critical path. For all these reasons, in this research, we focus on the setup error resolution in the three calculation FUs.

The detailed data-path auto-fix for a low power execution is carried out by three steps:

(1) Tuning under the low voltage. As shown in Section 3.1 and **Fig. 3**, we have given three possible voltage configurations for each array stage. The first attempt to fix the insufficient working frequency is to tune the three calculation operations inside the three execution units, under the `LOW` supply voltage configuration. As the operations inside a VLIW instruction do not have data dependence, they can change order freely. The function *re-map*`()` indicates the re-mapping control sequence by switching the instructions inside the three calculation units. After changing the mapping of the operations and their corresponding execution FUs, a new test—indicated as function *test*`()`—will be introduced to give a diagnose to check whether the new mapping satisfies the setup timing requirement. When all the three units give no timing error indication in the Razor-FF, this array stage records the current good mapping and a flag `tuned` is set for this stage to avoid a new tuning. Note that although the algorithm in Fig. 3 is written in `for (i=0;i≤N;i++)` style to explore all stages, the processing of this loop in hardware is actually done in parallel.

```
enum {ABC, ACB, BAC, BCA, CAB, CBA} ntune; /* six possible mappings */
enum {LOW, MEDIUM, HIGH} Vdd;

/* 1) Initialization */
for (i=0;i≤N;i++) /* Start from MEDIUM voltage */
    a[i].tuned = 0;
    /* If all tuning fail, Vdd = HIGH will be used to avoid error.  */
    a[i].good_Vdd = HIGH; a[i].good_map = ABC;

/* 2) Find optimized voltage for each stage */
for (Vdd=LOW;Vdd<HIGH;Vdd++) /* Exploring Vdd = LOW and Medium.  */
    for (ntune=ABC;ntune≤CBA;ntune++); /* Exploring six mappings.  */
        for (i=0,k=0;i≤N && k≤M;i++, k++)
            /* All stages, in parallel actually.  */
            if (a[i].tuned) continue;
            re-map(a[i], vliw[k], ntune); /* one cycle */
            if (a[i].tuned = test(a[i], Vdd)) /* two cycles */
                a[i].good_map = ntune; a[i].good_Vdd = Vdd;
```

Fig. 3 Tuning algorithm to find optimized voltage.

(2) Voltage up-scaling and tuning again. The second phase of this algorithm is to tune under the `MEDIUM` voltage. As can be expected, `LOW` voltage will still have a lot of setup timing violations when some VLIW instructions contain long critical path calculations which can not find a unit with a

sufficiently beneficial process variation. In the second phase, all the units is set to work with MEDIUM voltage which improves the switching speeds in the circuit. The voltage up-scaling is achieved by connecting each stage to the MEDIUM voltage network. Similar tuning procedures can be used for this tuning, except that the previously tuned unit will skip recording the new good tuning result, as programmed in Fig. 3.

(3) Finalizing. After the above two tuning procedures, the VLIW instructions that still can not find a good mapping to survive a low power execution mode will be applied to use HIGH supply voltage. Strictly speaking, under a modern or future process technology, it is possible to have large variations even under a high voltage setting. However, this should already be strictly reflected in the defined normal working frequency by adding sufficient margins. We assume that this HIGH voltage triggers far less setup timing errors than the low power mode and thus the tuning stops here. In the algorithm, we assign the good_Vdd = HIGH in the initialization part to reflect this step.

After these steps, the optimized voltage and mapping have been achieved in each array stage. We can now connect them to proper voltage networks and achieve a good power reduction without sacrificing any performance.

The time cost of each tuning can be composed of one cycle for re-mapping and two cycles for a new test. The two cycles of a test function is made up by a one-cycle dummy data execution and another cycle to collect Razor-FF comparison data. However, the second step of the test can be pipelined to overlap with the new re-mapping procedure. Accordingly, for six attempts of mapping, 13 cycles are respectively required for the LOW and MEDIUM voltages tunings. Compared to the waiting time for a stable state after a voltage up-scaling, this tuning by trying six combinations is relatively negligible. Also, we assume that LAPP works on a sufficiently large-iteration loop after each optimized mapping is set, so that all the tuning cost in this method is regarded as negligible in this paper.

3.2.4 Fixing Data-Path Vertically with a Single Voltage Network

Though it is possible to use multiply voltage networks in a single processor as shown in paper 14), it is generally argued that multiply voltage networks introduce unavoidable hardware and power cost. To avoid this, we also give another exploration to use only a single reduced voltage across the whole LAPP. The method is applied as follows.

```
enum {ABC, ACB, BAC, BCA, CAB, CBA, NIL} ntune;

/* Find optimized mapping */
i = 0; /* index of array stage */
k = 0; /* index of VLIW instruction */
while (k<M) /* to map all VLIW instructions */
    for (ntune=ABC;ntune≤CBA;ntune++); /* Exploring six mappings.  */
        re-map(a[i], vliw[k], ntune); /* one cycle */
        if (test(a[i], MEDIUM)) /* two cycles */
            a[i].good_map = ntune; a[i].tuned = 1; break;
    if (ntune>CBA) i++; continue; /* failed.  Try the next stage */
    k++; i++; /* found good map.  Try the next instruction */
```

Fig. 4 Tuning algorithm to find optimized map by skipping stages with adverse process variation.

As shown in **Fig. 4**, instead of tuning for an appropriately lowest stage for each VLIW instruction, we design to skip an array stage when it contains no sufficient beneficial process variation to support the current VLIW instruction, which is indicated by the i++ line in Fig. 4. The array stage is skipped by bypassing its input register values directly to the next stage. According to this minor complexity, the bypassed stage is granted to be free of setup timing errors. Eventually, when the VLIW instruction finds the beneficial process variation to hold it, the mapping of this instruction will be fixed, demonstrated as the setting of good_map in Fig. 4. The next instruction will be fetched and try to find its own proper array stage.

Accordingly, the mapping phase of all the VLIW instructions can not be performed in parallel and the mapping cost will be relatively high. However, in some applications where several cascaded loops are used, this possibly large mapping cost can be concealed, for the reason that only one mapping phase is necessary to fit the most inner loop kernel into LAPP.

In addition, another issue arises for this method, that a large array size may be necessarily required as part of stages will be bypassed. In Section 4, we will give a study of how many spare stages will be required in average.

3.3 Exploration: Possible Combination with Other Techniques

3.3.1 Core Fusions

Though this research has been carried out on LAPP which represents a CGRA-like reconfigurable FU array, it is possible to extend its application to the many-core architecture. The reason to use FU array is that this method requires a predetermined mapping inside the FU array. With this knowledge of the fixed data-path in the next execution phase, it is possible to tune via architectural ways to get a margin-free new mapping by fixing the data mapping. The many-core architecture is commonly used to exploit thread level parallelism, where our proposed method can not work. However, recently, new ideas such as different types of core-fusion[15),16)] are used to link a bundle of cores together to boost the sequential code execution. This is much like a coarse-grained FU array with a fixed data-path inside the bundle. Accordingly, similar methods like in this paper can be applied in the fused core to achieve margin-less execution, exploiting the beneficial process variations. Furthermore, it is easier to apply core-level different voltages without introducing large area and energy penalties. Paper 17) shows a method to map programs onto many-core architecture, under the profiling of process variations inside these cores. Compared to that method, our method uses a more dynamic way to get an optimized mapping, which can even give a positive usage of beneficial process variation.

3.3.2 Knowledge of Data Importance

Another promising direction to apply our method is to combine with explicit data importance indication. As introduced in papers 13), 18), data inside a program may have different importance. For example, when a loop is written as `for (i=0;i<MAX;i++) gray[i] = rgbtogray(r[i], g[i], b[i]);`, the control part, indicated as `i` and its calculation is more important than the rgb data and calculation, given that several pixel miscalculations can be tolerable. Accordingly, with a compiler to explicitly add indication of data importance, we can easily find an appropriate voltage and mapping for the control part and leave the other parts with less tuned mappings.

In some extreme cases, tuning of the rgb data part may not be necessary, as the indication of data importance has already given a less reliability requirement. However, by our tuning scheme, with a little bit more tuning, it is possible to improve significantly accuracy even with a same low power budget.

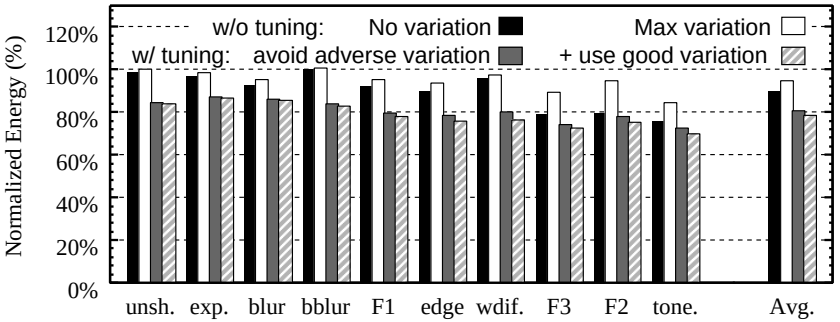

Fig. 5 Normalized energy results.

4. Results

In this section, we will show energy consumption results that we have achieved by using the proposed method. Ten loop kernels—mainly from image processing programs—are used for this research, as the baseline processor LAPP works best on this kind of programs which exploit high parallelism between loop iterations. The ten loop kernels are *unsharp*, *expand4k*, *blur*, *bblur*, *F1*, *edge*, *wdifline*, *F3*, *F2*, and *tonecurve*. Note that these loop kernels do not contain multiplication instructions so that we remove the multiplication unit from LAPP and use MEDIA operation as the baseline unit to define the frequency.

Fig. 5 shows the energy results of these loops before and after applying the proposed method. The horizontal axis gives the ten benchmarks, in a decreasing order of the averaged operation delay, i.e., *unsharp* tends to have more long delay instructions like MEDIA and *tonecurve* has more logic and shift operations. The vertical axis shows the energy consumption results, normalized by the baseline processor execution results. In order to give a clean study of the tuning method itself, the following experiment settings are given:

(1) The study is limited in the FU array part. DVS is not applied in other processor parts and their power consumption is not included in the results.
(2) Only mapped units consume power. Unmapped FUs in the array are power gated to show no impacts in the final results.

(3) 10^4 random variation settings are used and averaged as a stable result.

Four sets of experiments have been carried out, as shown in Fig. 5. The left most bar gives the possible energy reduction by aggressively applying Razor-FF and DVS, without including the process variation discussion. It represents the energy reduction by removing margins of system frequency over-design. In addition, margins of long delay units are also removed when they are not used. In average, 10.4% energy reduction can be achieved by this baseline use of Razor-FF.

However, when the maximum process variability is considered, as shown in the second bars in the loop kernels, it is very difficult to apply Razor and DVS. It is because the frequency is dominantly defined by the slowest unit in the processor. We assume Razor-FF can still help remove the margins of long delay units when they are not mapped. In average, only 5.60% energy reduction is possible in this experiment setting.

The third and fourth bars in each loop kernel represent the results of this paper. It clearly show that the margin included to avoid the possible process variation can be optimally removed by our tuning method. As shown in the third bars, if only adverse process variation is to be avoid, we can achieve an averaged energy reduction of 20.0%. Further, as also proposed in this paper, with the help of Razor-FF, it is also possible to go one step further to make good use of beneficial process variations. The results of using beneficial process variation are shown in the fourth bars in Fig. 5. In average, the beneficial process variation brings another 2.40%, as compared to the results of third bars. Accordingly, the total energy reduction achieved in this paper is 21.9% in average.

As introduced in Section 4, another way is to keep using a lowered supply voltage while skipping the array stage in case that the critical path requirement can not be fulfilled. **Fig. 6** shows the increased stage number when applying this method under an always MEDIUM supply voltage. Accordingly, the energy consumption is always 81% of the baseline processor. The proposed method achieves 19% energy reduction by a 16% of hardware size increase, in average. However, the required number of the array stages is defined by the longest loop kernel. Under this consideration, when the insufficiency of array stage happens, the fail-safe such as re-mapping under a global HIGH voltage is necessary.

As has been introduced, in this paper, MEDIA unit represents the unit with longest delay, and the program we are using are mainly media processing programs. These two reasons will partially limit the effectiveness of our method. In future work, the method will be carried out to benchmarks with more instructions of variable delays.

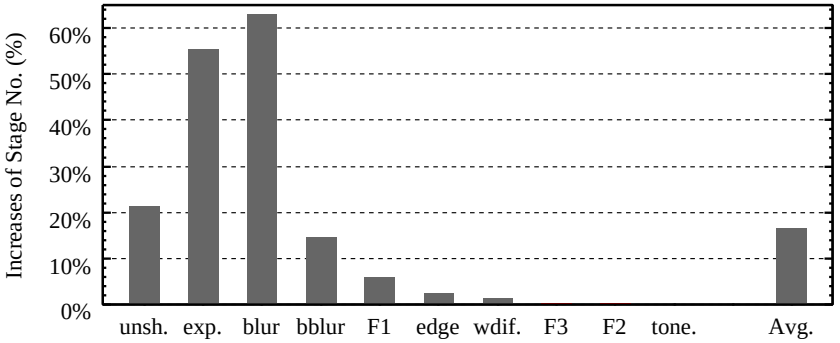

Fig. 6 The increase of No. of array stages by using Fig. 4.

5. Conclusion

In this paper, two methods based on an auto-fix of mapped data-path have been proposed to work with an FU array based processor. These methods are aware of process variability which becomes common with modern process technologies. Different to previous researches, we extensively use tuning ways to avoid using adverse process variations and in addition, detect and make good use of beneficial ones. Our results have indicated a 21.9% energy reduction is possible without sacrificing performance, even under a setting where power gating has been perfectly applied.

Acknowledgments This work is supported by VLSI Design and Education Center (VDEC), University of Tokyo with the collaboration of Synopsys Corporation. This work is supported by JST ALCA, JST A-STEP (F-S) No. AS232Z02313A, and Grant-in-Aid of No. 24240005, No. 24650020, and No. 23700060.

References

1) Mudge, T.: Power: a First-Class Architectural Design Constraint, *Computer*, Vol. 34, No.4, pp.52–58 (2001).
2) Li, H., Cher, C.-Y., Roy, K. and Vijaykumar, T.N.: Combined Circuit and Architectural Level Variable Supply-Voltage Scaling for Low Power, *IEEE Transactions on VLSI Systems*, Vol.13, No.5, pp.564–576 (2005).
3) Isci, C., Contreras, G. and Martonosi, M.: Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management, *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)*, pp.359–370 (2006).
4) Isci, C., Buyuktosunoglu, A. and Martonosi, M.: Long-Term Workload Phases: Duration Predictions and Applications to DVFS, *IEEE Micro*, Vol.25, No.5, pp. 39–51 (2005).
5) Sherwood, T., Perelman, E., Hamerly, G., Sair, S. and Calder, B.: Discovering and Exploiting Program Phases, *IEEE Micro*, Vol.23, No.6, pp.84–93 (2003).
6) Ernst, D., Kim, N.S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K. and Mudge, T.: Razor: a Low-Power Pipeline Based on Circuit-Llevel Timing Speculation, *Proceedings of 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003 (MICRO-36)*, pp.7–18 (2003).
7) Fuketa, H., Hashimoto, M., Mitsuyama, Y. and Onoye, T.: Transistor Variability Modeling and its Validation With Ring-Oscillation Frequencies for Body-Biased Subthreshold Circuits, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.18, No.7, pp.1118 –1129 (2010).
8) Alioto, M., Palumbo, G. and Pennisi, M.: Understanding the Effect of Process Variations on the Delay of Static and Domino Logic, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.18, No.5, pp.697 –710 (2010).
9) Ikeda, M., Ishii, K., Sogabe, T. and Asada, K.: Datapath Delay Distributions for Data/Instruction against PVT Variations in 90nm CMOS, *The 14th IEEE International Conference on Electronics, Circuits and Systems, 2007 (ICECS 2007)*, pp. 154 –157 (2007).
10) Kim, Y., Park, I., Choi, K. and Paek, Y.: Power-Conscious Configuration Cache Structure and Code Mapping for Coarse-Grained Reconfigurable Architecture, *Proceedings of the 2006 International Symposium on Low Power Electronics and Design (ISLPED'06)*, pp.310–315 (2006).
11) Yoshimura, K., Iwakami, T., Nakada, T., Yao, J., Shimada, H. and Nakashima, Y.: An Instruction Mapping Scheme for FU Array Accelerator, *IEICE Transactions on Information and Systems*, Vol.E94-D, No.2, pp.286–297 (2011).
12) Tiwari, A., Sarangi, S.R. and Torrellas, J.: ReCycle: Pipeline Adaptation to Tolerate Process Variation, *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*, pp.323–334 (2007).
13) Esmaeilzadeh, H., Sampson, A., Ceze, L. and Burger, D.: Architecture Support for Disciplined Approximate Programming, *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*, pp.301–312 (2012).
14) Hirai, K., Kato, M., Saito, Y. and Amano, H.: Leakage Power Reduction for Coarse-Grained Dynamically Reconfigurable Processor Arrays Using Dual Vt Cells, *The 2009 International Conference on Field-Programmable Technology (FPT'09)*, pp.104–111 (2009).
15) Ipek, E., Kirman, M., Kirman, N. and Martinez, J.F.: Core Fusion: Accommodating Software Diversity in chip multiprocessors, *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*, pp.186–197 (2007).
16) Venkatesh, G., Sampson, J., Goulding-Hotta, N., Venkata, S.K., Taylor, M.B. and Swanson, S.: QsCores: Trading Dark Silicon for Scalable Energy Efficiency with Quasi-Specific Cores, *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*, pp.163–174 (2011).
17) Ding, Y., Kandemir, M., Irwin, M.J. and Raghavan, P.: Adapting Application Mapping to Systematic Within-Die Process Variations on Chip Multiprocessors, *Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers (HiPEAC'09)*, pp.231–247 (2009).
18) Yao, J. and Nakashima, Y.: Exploiting Efficiency of Redundant Executions on an FU Array, *IPSJ SIG Notes*, Vol.2011-ARC-194, No.9, pp.1–5 (2011).