社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－ＡＲＣ－168（４）
2006／6／8

Dynamic Control Mechanisms for Pipeline Stage Unification Based on Program Phase Detection

Jun YAO† Hajime SHIMADA† Yasuhiko NAKASHIMA††
Shin-ichiro MORI‡ and Shinji TOMITA†

† Graduate School of Informatics, Kyoto Univ. 36-1 Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501 Japan
†† Graduate School of Information Science, NAIST 8916-5, Takayama-cho, Ikoma, Nara, 630-0192 Japan
‡ Graduate School of Engineering, Fukui Univ. 3-9-1 Bunkyo, Fukui, 910-8507 Japan
E-mail: †{yaojun, shimada, tomita}@lab3.kuis.kyoto-u.ac.jp
††nakashima@is.naist.jp, ‡moris@fuis.fuis.fukui-u.ac.jp

Abstract A method called pipeline stage unification (PSU) has been proposed to reduce power consumption for modern processors, which can work as an alternative for Dynamic Voltage Scaling (DVS) technology. In this paper, we proposed two mechanisms for PSU controller which can dynamically predicate a suitable unification based on the program phase detection. Our results show that the two mechanisms can achieve an average Energy Delay Product (EDP) decreasing of 15.1% and 19.2% for SPECint2000 benchmarks, compared to the processor without PSU.

Keyword Power Consumption, Program Phase Detection, Dynamic, Pipeline Stage Unification (PSU)

1. Introduction

To reduce power in modern processors, a method called dynamic voltage scaling (DVS) is currently employed. DVS reduces power consumption by decreasing the supply voltage while the processor is experiencing low work load. However, Shimada et al. [1, 9, 10] and Koppanalil et al. [2] have presented us a different method to reduce the processor power consumption via inactivating and bypassing the pipeline register and using a shallow pipeline during the program execution, which is called pipeline stage unification (PSU). PSU can save energy in the following ways:

1. Energy can be saved because of the clock gating of some pipeline registers.

2. After pipeline stage unification, a pipeline has fewer stages. Usually, a shallow pipeline will have better IPC due to decreased branch misprediction penalties and functional unit latencies compared to the deep pipeline, as illustrated in [3].

Our research described in this paper is focusing on how to control PSU hardware to achieve a good power saving. Currently there is only one research related to PSU control [11] and it mentions about execution with predefined throughput. In this paper, we propose some mechanisms to dynamically adjust the pipeline configuration to a suitable unification degree according to the program behavior change, so as to achieve better Energy Delay Product (EDP). By using the two different mechanisms described in this paper, we can get an average decreasing of 15.1% and 19.2% in EDP, respectively, compared with the EDP of processors under normal configuration. And compared these two mechanisms with unification degree 2, which usually have a good EDP efficiency, we can get a decreasing of 1.41% and 4.82%.

The rest of the paper is organized as follows. Section 2 describes the background techniques. Section 3 introduces the dynamic control mechanisms for a PSU enabled system. Simulation methodology and metrics to evaluate these mechanisms can be found in section 4. In section 5 we show the experiment results, together with some analysis. Section 6 concludes the paper.

2. Related works

2.1. Pipeline stage unification

Our work introduced in this paper is based on Shimada's previous PSU architecture, described in paper [1, 9, 10]. The PSU system can unify multiple pipeline stages by bypassing pipeline registers when the processor runs with low clock frequency.

Suppose the pipeline we are about to discuss will have 20 stages as shown in [1]. We assume 3 unification degrees in the latter part of this paper.

1. U1: The normal mode without bypassing any pipeline registers.

2. U2: Merge every pair of two adjacent pipeline stages

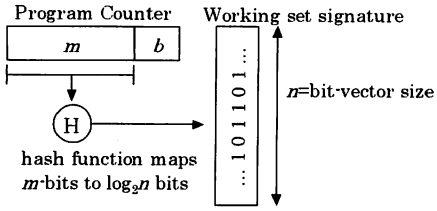

Fig. 1. Method for collecting working set signature

by inactivating and bypassing the pipeline register between them. The new pipeline will have 10 stages.

3. U4: Based on U2, merge the adjacent stages one step further. It becomes a 5-stage's pipeline.

2.2. Working set signature

Dhodapkar[4] and Sherwood[5] have shown that programs can be divided into phases in which program would have similar behaviors including the cache miss, IPC and power consumption.

In order to detect the phase changes during the program execution, Dhodapkar designed a working set signature to work as the compacted representation for an instruction interval. Figure 1 shows the method to form a working set signature. "*b*" in Figure 1 is the number of bits which are used to index a instruction in one cache block. During the execution, Dhodapkar selected *m* bits from the program counter and used these bits to address 1 bit in the n-bit signature via a hash function. The signature is cleared at the beginning of an instruction interval. After the interval begins, a bit in signature is set if the corresponding instruction cache block is touched. Dhodapkar used a 1024-bit signature in his paper. Each instruction interval has 100k instructions. The hash function he described is based on the C library srand and rand.

In paper [4], the distance δ between two signatures S1 and S2 is calculated as follows:

$$\delta = \frac{num\_of\_"1"bit\_in(S_1 \oplus S_2)}{num\_of\_"1"bit\_in(S_1 + S_2)} \quad (1)$$

Where num_of_"1"bit_in() represents the function that counts the number of "1" bits in the bit vector. If the distance δ is larger than a predefined threshold delta, the two instruction intervals are of different program phases. Dhodapkar used 0.5 as threshold in his paper.

3. Dynamic PSU control mechanisms

Based on Section 2, we can make such assumptions that since the energy consumption keeps nearly flat in a stable program phase, we can use the same pipeline stage unification degree in that phase and tune a new pipeline

```
After each interval I_k:
δ =signature distance of I_k and I_{k-1};
if (state == stable)
   if (δ> threshold)
      state = unstable;
      unification_degree = U1;
else if (state == unstable)
   if (δ<=threshold)
      state = tuning;
      unification_degree = U1;
else if (state == tuning)
   if (δ>threshold)
      state = unstable;
      unification_degree = U1;
   else if (unification_degree == U4)
      state = stable;
      unification_degree=best from tuning;
```

Fig. 2. Algorithm of basic phase detection method

stage unification degree at the phase switching point. In the following section, we applied 2 different control methods on the original PSU system: basic phase detection method and history table based method.

3.1. Basic phase detection method

We got the idea from Balasubramonian et al.[6], and changed the algorithm a bit to work with the PSU system. The algorithm is shown in Figure 2.

We have three states in this algorithm:

1. stable: The adjacent intervals are of same phases;

2. unstable: A phase switching in program occurs and current interval is of different phase with last interval;

3. tuning: The period when the adjacent intervals become stable again and different unification degrees are being explored.

After each program interval, we compare the signature of current interval with the signature of the previous interval. If the distance is larger than the threshold, we change the state to unstable. For simplicity, we use U1 as the unification degree for the unstable phase. The next intervals are unstable until the distance becomes smaller than the threshold again. Then we change the state to tuning, which tries different unification degrees in the following three intervals and collect the corresponding EDP. After tuning, if the interval is still under the same program phase, we can choose a best unification degree for this phase and set the state to stable. This algorithm is based on the assumption that program will show same

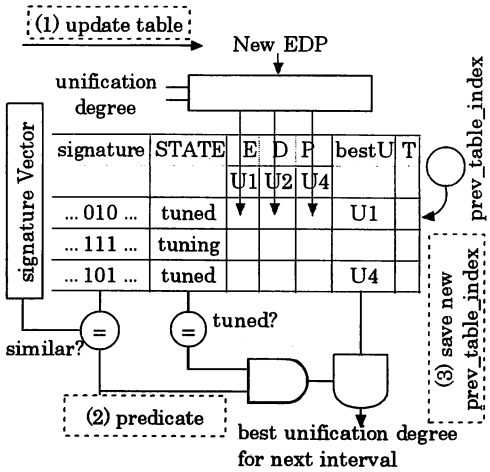

Fig. 3. hardware approach for history table based method

behavior including energy, performance and so on in the same program phase.

Because we only compare the signatures of each consecutive interval pair, this method is of low cost. The corresponding control hardware will also show the advantage of simplicity.

3.2. History table based method

We designed the table based method to keep the phase information in a history table. If the program comes into a phase that has appeared in the past, we can choose a suitable unification degree from the cached history information without starting a new tuning procedure.

Figure 3 is the diagram of the hardware approach of this table method; Figure 4 shows the detailed algorithm of history table based method.

The table that we are using in this algorithm is constructed in the following way:

1. The signature field: Each different signature occupies one table entry so that we can use this field to index the table items.

2. The state field: It denotes the state of the table entry. We define two states here: tuned and tuning. A state of tuning means that the best unification degree is still not figured out. After all three unification degrees have been tried, we select a best unification degree from the tuned EDP results (another field in the table) and set the state as tuned. 1 bit is used for this field.

3. The EDP field occupies three fixed point storage units for each entry. We keep the tuning information of different unification degrees in the three fields denoted as U1, U2 and U4, respectively. They are updated when the

```
After each interval I_k:
if (prev && prev->state==tuning)
   prev->EDP[unif_degree]=EDP for I_k;
   if (unif_degree==U4)
      prev->bestU=best(prev->EDP[U1, U2, U4]);
      prev->state = tuned;
v = find_nearest_signature();
δ = signature distance between v->sig and I_k;
if (!v || δ>threshold) /* miss */
   v=new_table_entry();
   v->sig=signature of Ik;
   unif_degree = U1;
   v->state = tuning;
else if (v->state == tuned)
   unif_degree = v->bestU;
else /* v->state == tuning */
   unif_degree = next unif_degree for v;
prev = v;
```

Fig. 4. Algorithm of history table mode

entry is under the tuning state.

4. The bestU field: It holds the best unification degree for this signature. This field is set after the tuning finishes. If this phase occurs again, we can use the suitable unification degree from this field. Two bits are used for this field.

5. The T field: It records the time that the entry is touched. We use it when replacing old entries. Several bits are used according to the table size.

At the point that we are about to predicate a suitable unification degree for next interval "I_{k+1}", it does not really start so that we do not know the signature. Hence we have no index to look up the history table and can hardly predicate the best unification degree. To solve this problem, we engage a specific register named prev_table_index to store the table index of the previous interval. After each interval, we calculate the EDP of current interval and store it in the entry which prev_table_index refers to ((1) in Figure 3). Therefore the EDP field and best unification degree field of each current entry hold the information for the next interval. After current interval finishes, we can look for the current signature in the history table. If there is a hit, the corresponding entry will probably carry the best unification degree for the next interval. And we can predicate the best unification degree based on this entry ((2) in Figure 3). The register prev_table_index will be

updated to current table index before we start the next interval ((3) in Figure 3).

In Figure 4, "prev" denotes prev_table_index and "v" denotes a temporary table index. Also the syntax like "prev->state" denotes the "state" field of the entry point by prev. unif_degree is the current unification degree.

There are two main actions for the table in Figure 4:

1. Find the nearest signature. Look up the table and find a signature that has the smallest distance with current one;

2. Replace the least recently used table entry when there is no sufficient place for the coming new signature.

The performance of these two actions will greatly depend on the size of the table. Detailed results about the table size will be given in Section 5.4.

In this method we have an assumption that if interval I_{k+1} once happens after interval I_k and I_k occurs again, the next interval will probably be I_{k+1}. It is a bit like a simple history branch predicator. We can efficiently predicate the best unification level for I_{k+1} if the next interval for I_k is always I_{k+1}, while we must endure some mispredication penalty if the next interval for I_k is variable. We will show the efficiency of this method in Section 5.

4. Simulation methodology

We use a detailed cycle-accurate out-of-order execution simulator, Simplscalor Tool Set [7], to measure energy and performance of different unification degrees. Table 1 lists the processor configuration. We assume a deep pipeline similar to the current processors. Table 2 summarizes the latencies and penalties in pipeline configuration of U1, U2 and U4, respectively.

We used 8 integer benchmarks (gzip2, gcc, gzip, mcf, parser, perlbmk, vortex and vpr) from SPECint2000, with train inputs. 1.5 billion Instructions are simulated after skipping the first billion instructions.

To evaluate the energy and performance together in the tuning procedure, we can use PDP, EDP and EDDP as the metric, which can be calculated as W/MIPS, $W/(MIPS)^2$ and $W/(MIPS)^3$, respectively [8]. For simplicity, we apply one single metric during one program execution. The experiments and analysis in Section 5 are based on EDP because our PSU is targeted on high-performance mobile computer. Our mechanisms can easily change to the metric of PDP or EDDP to fit for different platforms.

In this paper, we calculate the energy saving in the processor. We get eq.2 from paper [1, 9, 10] to calculate the energy saving under different unification degrees.

Table 1: processor configuration

Processor	8-way out-of-order issue, 128-entry RUU, 64-entry LSQ, 8 int ALU, 4 int mult/div, 8 fp ALU, 4 fp mult/div 8 memory ports
Branch predication	8K-entry gshare, 6-bit history, 2K-entry BTB,16-entry RAS
L1 I cache	64KB/32B line/2 way
L1 Dcache	64KB/32B line/2 way
L2 unified cache	2MB/64B line/4-way
Memory	64 cycles first hit, 2 cycles burst interval
TLB	16-entry I-TLB, 32-entry D-TLB, 128 cycles miss latency

Table 2: Assumptions of latencies and penalty

unification degree	U1	U2	U4
clock frequency rate	100%	50%	25%
branch mispredication penalty	20	10	5
L1 Icache hit latency	4	2	1
L1 Dcache hit latency	4	2	1
L2 cache hit latency	16	8	4
int Mult latency	3	2	1
fp ALU latency	2	1	1
fp Mult latency	4	2	1

Where E_{Ux} is energy in unification degree Ux and Enormal is energy in normal execution mode; IPCnormal is IPC in normal execution while IPC_{Ux} is IPC in Ux; β is the power saving part from inactivated pipeline registers. Since half of the pipeline registers are inactivated in U2, we can get a β of 15%. Furthermore, for U4, an extra half of pipeline registers are inactivated, we can get a new β of 22.5%, as described in paper [1].

$$\frac{E_{Ux}}{E_{normal}} = \frac{IPC_{normal}}{IPC_{Ux}} \times (1-\beta) \qquad (2)$$

5. Results and analysis

5.1. Two non-phase based methods for comparison

Before applying our algorithms on the PSU controller, we have another two methods which are used to measure the efficiency of the phase detection based algorithms.

(1) Single unification degree method

Use a fixed unification degree U1, U2 or U4 in the whole execution and calculate EDP data of each interval.

(2) Optimal method

Based on the data collected from single unification degree method, we can find a best unification degree for each interval. By using such profiling data we can set the unification degree to the best one at the beginning of each instruction interval. This method is a theoretical optimal one. If the EDP result of another mechanism is close to

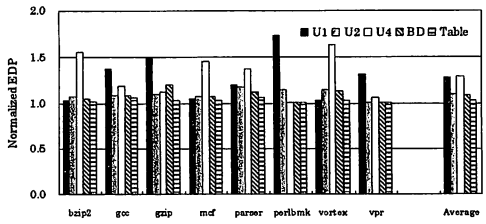

Fig. 5. Normalized EDP for SPECint2000 benchmarks

this optimal one, we can say that mechanism is efficient.

5.2. General analysis via comparing average EDP

We chose the signature size to be 1024 bits and the threshold to be 0.5. Each interval has 100k instructions. A simple hash function based on division is used to introduce low cost in the signature collection. Figure 5 shows the EDP results for all 8 benchmarks. In Figure 5, the horizontal axis denotes benchmarks and the average value, and the vertical axis denotes EDP value normalized by EDP of the optimal method for each benchmark. The columns in one benchmark represent the normalized EDPs of U1, U2, U4, basic phase detection and history table based method, from left to right. Also the average results of all benchmarks are listed. The method of smaller EDP result is more efficient.

For the efficiency of our mechanisms, Figure 5 shows that the basic phase detection method can achieve an average EDP of 108%, compared with the optimal method. And it obtains a decreasing of 15.1%, 1.41% and 16.4% when compared with single U1, U2 and U4, respectively.

The history table based method shows better average results, as compared with the basic phase detection method. It can achieve an average EDP of 103% of the optimal method. Compared with single U1, U2 and U4, it can gain a total EDP decreasing of 19.2%, 4.82% and 20.5%.

5.3. Predication accuracy

Since we are designing the dynamic mechanisms to predicate a suitable unification degree for the next interval, the predication accuracy is very important to the final energy saving result. We list the predication accuracy of the unification degrees in Table 3, together with some benchmark characteristics. In Table 3, the column of Stable Rate stands for the percentage of the total intervals that are in stable time. The "nSigs" column denotes the number of different signatures when the programs are under the stable time. A higher value shows

Table 3. Predication accuracy of each benchmark, together with benchmark characteristics.

Bench-mark	Stable Rate (%)	nSigs	Avg. ST_Len.	Pred. Acc.(%)	
				BD	Table
bzip2	86.80	12	28.93	85.78	94.73
gcc	89.73	55	53.84	60.77	51.95
gzip	59.18	3	9.575	66.58	84.68
mcf	32.98	6	2.382	41.30	49.80
parser	67.63	33	11.75	49.24	60.12
perl.	99.97	1	14995	99.98	99.98
vortex	51.74	6	4.720	46.16	87.41
vpr	99.97	1	14995	99.98	99.98

that the programs can be classified into more different stable phase groups and may require more tunings. The column of "Avg. ST_len" represents the average interval length of the stable phase.

The column named "Pred. Acc." in Table 3 is the ratio of precise predication of the unification degree for the two dynamical methods, respectively. We got these two columns by calculating the similarity of predicated unification degrees with those theoretical precise unification degrees from optimal method.

Generally, from Table 3, we can see that the predication accuracy of table based method is better than the basic detection based method. This is similar with the conclusion we have obtained in Section 5.2.

Also from this table, we can see that the predication accuracy changes due to the program characteristics. For some simple benchmarks like perlbmk and vpr, most intervals are of the same stable phase and the predication accuracy of both dynamical methods can reach nearly 100%. The accuracy of the basic detection based method drops visibly when the program becomes less stable. This may related with its simple design. We only compare the signatures of consecutive intervals and start a tuning at each point the program goes toward stable. If the program stability is low, this method will get hurt since energy can hardly be saved in unstable and tuning phase.

Different with basic detection method, history table base method is less sensitive to the program stability. It is well illustrated from benchmarks like gzip and vortex. Although the stability ratios for these two benchmarks are lower than 60%, the predication accuracy can still reach 84.68% and 87.41%, respectively. This is because the table based method records the historical tuned information in extra structures. If the jump direction from one signature to another signature is stable, the predication will be accurate. But on the other hand, this method is sensitive to the number of phase groups. For

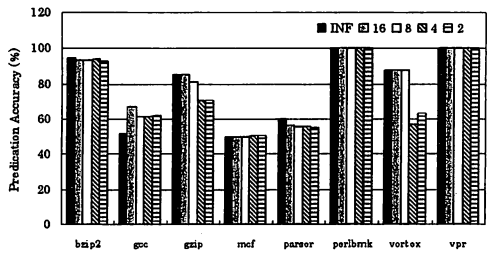

Fig. 6. Predication accuracy of different table size for table based method

example, gcc is quite stable but the number of different signatures is large, which leads to the uncertainty in the jump directions and hurts the accuracy. More detailed results of table based method for gcc will be listed in Section 5.4.

5.4. Table size

The size of the history table is another important parameter for the table based method. In this serial of experiments, we set the table size as a fixed number, from 16 entries to 2 entries. We use the LRU mechanism to replace the table entry when there is no sufficient place for new signature. The results are shown in Figure 6.

In Figure 6, the columns for each benchmark represent the results of infinite table size, 16-entry, 8-entry, 4-entry and 2-entry, respectively. We can see from the results that there is nearly no degradation between the infinite-entry, 16-entry and 8-entry for all benchmarks. A sharp decreasing of predication accuracy occurs on 4-entry table size for gzip and vortex. And for benchmark gcc, the accuracy even increases after we reduce the table size from infinite to 16 entries. The results of 8-entry and 4-entry are also better than the infinite one. It seems that for gcc, the old history information may sometimes have a bad impact in helping the predication.

From these results, we can assume that an 8-entry table size will be sufficient for SPECint2000 benchmarks. With a small table size, we can look up the table faster so as to introduce less overhead into the PSU control system.

6. Conclusions and future work

In this paper, we have designed two dynamic control mechanisms for PSU enabled processors in order to achieve good EDP. These two mechanisms are based on phase detection via working set signature. By using these two methods, we can dynamically reconfigure the unification degree during the program execution due to the program behavior change. Our simulation show that the two methods can achieve an average EDP decreasing of 15.1% and 19.2%, compared to the original system without PSU enabling. Such results are about 8.34% and 3.02% larger than the optimal mode. The basic detection method is simple and introduces less hardware complexity, while the history table based method shows better efficiency in predicating.

Currently the energy consumption model in this paper is still very rough. We are planning to study the hardware approach so as to build a more accurate model, including the detailed overhead introduced by the dynamical predication mechanisms. Also, different program phase detection methods other than the working set signature will be tried on the PSU system.

Acknowledgement

This research is partially supported by Grant-in-Aid for Fundamental Scientific Research (S) #16100001 from Ministry of Education, Culture, Sports, Science and Technology Japan.

References

[1] H. Shimada, et al., "Pipeline Stage Unification: A Low-Energy Consumption Technique for Future Mobile Processors", ISLPED2003, pp. 326-329, Aug. 2003.

[2] J. Koppanalil, et al., "A Case for Dynamic Pipeline Scaling", CASES2002, pp. 1-8, Oct. 2002.

[3] M.S. Hrishikesh, et al., "The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays", ISCA29, pp. 14-24, May 2002.

[4] A.S. Dhodapkar and J.E. Smith, "Managing Multi-Configuration Hardware via Dynamic Working Set Analysis", ISCA29, pp. 233-244, May 2002.

[5] T. Sherwood, et al., "Discovering and Exploiting Program Phases", IEEE Micro, Vol. 23, No. 6, pp. 84-93, Nov.-Dec. 2003.

[6] R. Balasubramonian, et al., "Memory Hierarchy Reconfiguration for Energy and Performance in General Purpose Architectures", MICRO33, pp. 245-257, Dec. 2000.

[7] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, version 2.0. Technical Report", CS-TR-97-1342, Univ. of Wisconsin-Madisoon Computer Sciences Dept., 1997.

[8] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors", IEEE JSSC, Vol. 31, No. 9, pp. 1277-1284, Sep. 1996.

[9] H. Shimada, et al., "Pipeline with Variable Depth for Low Power Consumption (in Japanese)", IPSJ Technical Report, 2001-ARC-145, pp. 57-62, 2001.

[10] H. Shimada, et al., "Pipeline Stage Unification for Low-Power Consumption," Cool Chips V, pp. 194-200, Apr. 2002.

[11] H. Shimada, et al., "A Hybrid Power Reduction Mechanism Using Pipeline Stage Unification and Dynamic Voltage Scaling (in Japanese)", SACSIS2004, pp. 11-18, May 2004.