Power-Directed Software Prefetching Algorithm with Dynamic Voltage Scaling*

Juan Chen, Yong Dong, Huizhan Yi, and Xuejun Yang

School of Computer, National University of Defense Technology, P. R. China
{juanchen, yongdong, huizhanyi, xjyang}@nudt.edu.cn

Abstract. We first demonstrate software prefetching provides an average 66.28% performance enhancement with much higher average power on six memory-intensive benchmarks. Then we propose a power-directed software prefetching algorithm with dynamic voltage scaling (*PDP-DVS*) that monitors a system's power and adapts the voltage level accordingly to guarantee no power increase while maintaining good performance boost. Our *PDP-DVS* algorithm achieves a 35.75% performance gain with only 1.19% power increase.

1 Introduction

High power consumption has become an important limiting factor in developing designs for battery-operated embedded systems due to exorbitant cooling, packing and power costs. Unfortunately, some traditional compiler optimization techniques are only aimed at improving program performance, which causes significant power increase. For example, software prefetching[1] improves the performance by overlapping CPU computing and memory access operation. However, inserting prefetching instruction and overlapping computing operation and memory access increase the probability of processor unit utility, which leads to power increase as Fig. 2 and Fig. 3 show. In this paper, we propose a power-directed software prefetching algorithm with dynamic voltage scaling (*PDP-DVS*), which eliminates power increase due to software prefetching while obtaining significant performance enhancement.

Agarwal et al. [2] presented a similar work, but their objective is to reduce energy consumption without performance loss. Furthermore, our *PDP-DVS* algorithm uses selective prefetching method besides DVS method.

2 Power-Directed Software Prefetching

We use the algorithm developed by Mowry[1] for prefetching affine array accesses and indexed array references. Let P_p and P_{np} be the average powers to run one part of codes with and without prefetching, respectively. To reduce power from P_p to P_{np} , the

* This work was supported by the Program of Nature Science Fund under Grant No. 60633050 and was supported by the National High Technology Development 863 Program of China under Grant No. 2002AA1Z2101 and No. 2004AA1Z2210.

optimal way to is to reduce voltage to v^* from v such that $\frac{P_p}{P_{np}} = \frac{v(v-v_l)^\alpha}{v^*(v^*-v_l)^\alpha}$ with respect to $P \propto C v^2 f$ and $f \propto (v-v_l)^\alpha / v$.

```

1. ALGORITHM: PDP-DVS
2. INPUT: A program with prefetching, power without prefetching (objpower)
3. OUTPUT: A program with real time voltage scaling
4.  $v = V_{max}$ ;
5. repeat for each  $N$  instructions till the completion of the application {
6.   execute  $M$  instructions with prefetching at  $v$ ;
7.    $P_p$ =average power for the execution of these  $M$  instructions;
8.   execute the next  $M$  instructions without prefetching at  $v$ ;
9.    $P_{np}$ =average power for the execution of these  $M$  instructions;
10.   $T_{np}$ =execution time for the execution of these  $M$  instructions;
11.   $ratio = P_p/P_{np}$ ;
12.  if ( $ratio \leq 1$ )
13.    execute the rest instructions with prefetching at the current voltage  $v$ ;
14.  else {
15.    calculate new voltage  $v^*$  such that  $ratio = \frac{v(v-v_l)^\alpha}{v^*(v^*-v_l)^\alpha}$ 
16.    execute the next  $M$  instructions with prefetching at  $v^*$ ;
17.     $P_{lp}$ =average power for the execution of these  $M$  instructions;
18.     $T_{lp}$ =execution time for the execution of these  $M$  instructions;
19.    if ( $T_{lp} > T_{np}$ )
20.      execute the rest instructions without prefetching at  $v$ ;
21.    else
22.      execute the rest instructions with prefetching at  $v^*$ ;
23.  }
24.  if ( $avepower > objpower$ )
25.    voltage_down();
26.  else
27.    voltage_up();
28. }
```

Fig. 1. Pseudo code for the profiling DVS algorithm PDP-DVS

Fig. 1 illustrates our power-directed software prefetching algorithm with DVS (PDP-DVS). This algorithm periodically conducts periodic profiling to estimate the average power increase due to prefetching and eliminate such power increase by DVS and selective prefetching.

For each N instructions (called **repetitive period**), we execute the first M instructions with prefetching and the next M instructions without prefetching. The latter can be achieved by treating prefetch instructions as NOPs. Assume that they take (P_p) and (P_{np}) of average power dissipation, respectively and take T_{np} of execution time without prefetching. If the ratio P_p to P_{np} is no more than 1, prefetching does not bring power increase, and we remain the voltage unchanged to execute the remaining instructions with prefetching (steps 12-13). We call executing these remaining instructions as **profile-guided period**, which is equal to $(N-2*M)$ instructions and we call the previous two M instruction as **profiling period**. That is, each repetitive period consists of profiling period and profile-guided period.

When we identify power increase, we execute the next M instructions with prefetching at lower voltage v^* . Note in this case, profiling period includes $3*M$ instructions instead of $2*M$ instructions. Assume the average power for the third M instructions is P_{lp} and execution time is T_{lp} . $T_{lp} > T_{np}$ represents prefetching cannot bring any performance boost. In this case, non-prefetching is better than prefetching and we will give up prefetching for the remaining instructions while remaining the previous operating voltage v (steps 19-20). We scale voltage up only when cumulative

Fig. 2. Power increase due to prefetching

Fig. 3. Execution time reduction due to prefetching

Benchmark	Power Increase by DVS	Time Reduction by DVS	Power Increase by SVS	Time Reduction by SVS
RB	4.92%	37.61%	25.81%	44.66%
JACOBI	4.74%	43.16%	26.91%	49.60%
MATMULT	0.53%	43.89%	28.39%	52.80%
IRREG	0.37%	30.93%	22.64%	39.61%
MOLDYN	-0.24%	36.47%	30.75%	46.22%
NBF	-3.16%	22.44%	14.94%	30.29%
Average	1.19%	35.75%	24.91%	43.86%

Table 1. Power and Performance Gain Achieved by DVS and SVS

average power (*avepower*) is lower than the objective power (*objpower*) (steps 26-27). Here objective power is the power for non-prefetching version. We use the following parameters during our simulation: $N=100k$ instructions (repetitive period), $M=5k$ (profiling period) and *objpower* (power for non-prefetching version).

Our experiment uses the six benchmarks, among which the first three applications perform affine array accesses and the others perform indexed array accesses. The description for all the benchmarks can be found in [3].

We use Wattch[4] to implement our *PDP-DVS* algorithm. Prefetch instruction was added to the ISA of the processor model and our simulation is based on $0.1\ \mu m$ process technology parameters with ideal clock gating. We use 800MHz and 1.65V as the baseline frequency and voltage. And the relationship between frequency f and voltage v meets: $f \propto (v - v_t)^\alpha / v$, where $\alpha=2$.

A suitable voltage level for the whole program according to the average power increase ratio is commonly not exact enough because each part of the program assumes different power consumption. From Table 1, the power dissipation after such static voltage scaling is still 24.91% larger than the original version. In contrast, *PDP-DVS* algorithm obtains 35.75% performance gain (time reduction) with only 1.19% power increase in average. Power increase ratio and time reduction ratio both take the non-prefetching version as a basis. For RB and JACOBI, our online *PDP-DVS* algorithm causes more power increase than other applications because RB and JACOBI have much less repetitive periods (only 62) than other applications.

Fig. 4 gives the dynamic voltage scaling results throughout a complete run of two applications. The others have the similar curves. During our online *PDP-DVS*, there isn't a steady optimal voltage at which system power can eliminate power increase completely. Instead, the optimal voltage steadily fluctuates within a range as Fig. 4 shows. In this Figure, horizontal axis represents number of repetitive periods. Vertical axis represents the optimal operating voltage for each profile-guided period.

In theory, ideal static operating voltage should lie in the midst of the peak value and lowest value of wave. However, the voltage setting by SVS is always higher this ideal value. We can see this from Fig. 4, where a red straight line represents static voltage scaling results. For MOLDYN, static voltage setting is almost at the peak value of wave so that its power increase ratio by SVS is the biggest in Table 1.

Take RB as an example, we notice each repetitive period almost includes the same number of prefetch instructions as Fig. 5 shows. Fig. 6 shows the average power of each repetitive period before applying *PDP-DVS* algorithm. One can see the average power of each repetitive period has periodic varying trend, which is different from cumulative power. **Cumulative power** at one point is the ratio of current total energy to the whole execution time as Fig. 7 shows. At the end of application execution, cumulative power is equal to the average power of the whole program. After using *PDP-DVS* algorithm, cumulative power reduces greatly at first, and fluctuates around 1W as Fig. 8 shows.

In our algorithm *PDP-DVS*, selective prefetching is used (steps 15-22 in *PDP-DVS*). Fig. 9 gives the selective prefetching profiles for RB. Due to space limitations, we omit the other applications' profiles, which have similar results. In Fig. 9, vertical axis represents selecting prefetching or not for each repetitive period, where "1" represents prefetching and "0" represents non-prefetching. Take RB as an example, there are 35 repetitive periods selecting non-prefetching among all 62 repetitive periods. Selective prefetching result for RB is simply denoted as 35/62. Similarly, 28/62 for

Fig. 4. Dynamic voltage scaling results by *PDP-DVS* algorithm and static voltage scaling results

Fig. 5. The number of prefetch instructions for each repetitive period for RB

Fig. 6. Average power of each repetitive period before using *PDP-DVS* algorithm for RB

Fig. 7. Cumulative power at the beginning point of each repetitive period before using *PDP-DVS* algorithm for RB

Fig. 8. Cumulative power at the beginning point of each repetitive period after using *PDP-DVS* algorithm for RB

Fig. 9. Selective prefetching profiles of *PDP-DVS* algorithm

JACOBI, 351/669 for MATMULT, 299/556 for IRREG, 1018/1858 for MOLDYN and 231/413 for NBF.

References

- [1] Todd C. Mowry. Tolerating Latency Through Software-Controlled Data Prefetching. Ph. D. thesis. Stanford University. Computer System Laboratory. March 1994.
- [2] Deepak N. Agarwal, Sumitkumar N. Pamnani, Gang Qu and Donald Yeung. Transferring Performance Gain from Software Prefetching to Energy Reduction. In *Proceedings of the 2004 International Symposium on Circuits and Systems*. Vancouver, Canada. May 2004.
- [3] Abdel-Hameed Badawy, Aneesh Aggarwal et al. The Efficacy of Software Prefetching and Locality Optimizations on Future Memory Systems. *Journal of Instruction-Level Parallelism*, .2004.
- [4] D. Brooks, V. Tiwari and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of 27th International Symposium on Computer Architecture*. June 2000. p83-94.