12.2 Dynamic Microarchitecture Adaptation via Co-Designed Virtual Machines

James E. Smith, Ashutosh S. Dhodapkar
{jes, dhodapka}@ece.wisc.edu
Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI

As microarchitecture and circuit technologies evolve, tradeoffs involving performance, power, and design complexity become increasingly difficult, and optimization methods become increasingly sophisticated. An important next step is toward microarchitectures that dynamically adapt to changing program characteristics. Researchers have put forward several proposals for multi-configuration hardware subsystems targeted at performance and/or power optimization.

1. *Caches and TLBs*: Total size, line size, and ways can be configured dynamically to match program requirements. Power can be saved by using the smallest structures that give adequate performance.

2. *Branch predictors*: Global history length can be varied to optimize performance. Powering down unneeded portions of a complex predictor can save power.

3. *Issue windows and pipelines:* To save power, sections of the instruction issue window can be shut down when there is a low instruction parallelism. Portions of clustered microarchitectures can be disabled when not needed.

Simultaneously managing several such multi-configuration units is a complex optimization problem. To manage this complexity, we are developing a co-designed virtual machine (VM), a layer of software hidden from all conventional software and designed concurrently with the hardware implementation. The base technology is used in the Transmeta Crusoe and the IBM Daisy projects to support whole-system binary translation. Referring to Fig. 12.2.1, we use the virtual machine monitor (VMM) as a "micro-operating system" for managing configurable resources in the microarchitecture.

A key aspect of configuration management is matching the configuration with the requirements of a program's current phase of execution. The phase, in turn, is a manifestation of the program's instruction *working set*. Consequently, we are interested in *detecting* working set changes and in *identifying* recurring working sets. A working set change indicates that the VMM should determine the optimal configuration for the new working set. Identifying recurring working sets enables the VMM to re-instate a previously determined optimal configuration. Fig. 12.2.2 shows pseudo code for a reconfiguration algorithm to be used by a VMM.

We have developed a simple hardware mechanism that the VMM uses to detect instruction working set changes and identify recurring working sets. The working set is captured by hashing the addresses of executed conditional branch instructions into a table holding a bit vector (Fig. 12.2.3). When a table entry is touched, the corresponding bit is set to one. Instructions are sampled for a fixed interval of time, and at the end of the interval, the bit vector *signature* is compared with the one generated during the previous interval.

For comparing two signatures we have developed a measure of "similarity". The total number of ones in the exclusive OR of the signatures is divided by the total number of ones in their inclusive OR. This *working set ratio* indicates the degree to which the signatures differ. If the two working set signatures are very similar, the ratio is close to zero; if they are very different, the ratio is close to one. In practice, if the working set ratio is more than some threshold, a working set change (phase change) is registered; if not, the phase is stable. We performed a number of simulations using SPEC 2000 benchmarks and found that a threshold of .125 works well. Fig. 12.2.4 tabulates the statistics gathered by the phase table for a subset of the SPEC 2000 benchmark suite.

To evaluate the effectiveness of using working set signatures, we performed a preliminary study that compares a working set signature-based method and a phase table based method with an "oracle" algorithm. Level one instruction and data caches can be independently adjusted to 2K, 8K and 32K byte sizes and a *gshare* branch predictor can be configured into 1K, 2K and 4K entries. This gives a total of 27 possible configurations. We define an *optimal* configuration during a given working set interval (100K instructions) as one which provides performance within 2% of the maximum configuration and which minimizes power consumption. The oracle algorithm is applied at the end of every working set sampling interval and performs reconfiguration when the performance falls outside the 2% optimal range. The signature-based algorithm reconfigures only when a working set change is detected due to a working set ratio of .125 or more. The phase table based algorithm goes one step further and simply looks up the optimal configuration from a table, if the phase had occurred in the past. The performance simulations use a 4-way superscalar processor model. Power consumption for the caches and predictor were generated using the Wattch power estimation tool (developed at Princeton University). Fig. 12.2.5 shows the performance achieved using each of the schemes. Fig. 12.2.6 shows 1) the power saved relative to the base case using each of the algorithms and 2) the ratio of number of reconfigurations performed by the oracle algorithm to that performed by the signature based algorithm. Both the signature-based algorithm and the phase table based algorithms achieve performance and power savings similar to the oracle algorithm, thereby suggesting that the signature mechanism detects most significant working set changes and identifies recurring phases correctly. The overall power savings for the caches and branch predictor range from 10% to 60%. Furthermore, the number of re-configurations is lower than the oracle method by an average factor of 20.

As we move towards increasingly complex microarchitectures, power/performance constraints will necessitate dynamic adaptation. Co-designed virtual machines are excellent candidates for managing the reconfigurable hardware, providing rich functionality and flexibility to the hardware designer.

Acknowledgements

This work is being supported by an SRC grant 2000-HJ-782, NSF grant CCR-9900610, Intel and IBM.

References

[1] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General Purpose Architectures," 33rd *Intl. Sym. on Microarchitecture*, pp. 245-257, Dec. 2000.

[2] T. Juan, S. Sanjeevan, J. Navarro, "Dynamic History-Length Fitting: A Third Level of Adaptivity for Branch Prediction," *25th Intl. Sym. on Comp. Architecture*, pp. 155-166, Jul 1998.

[3] H. Mizuno, T. Kawahara, "ChipOS: Open Power-Management Platform to Overcome the Power Crisis in Future LSIs", *ISSCC 2001*, pp. 344-45.

[4] K. Ebcioglu, E. Altman, "DAISY: Dynamic Compilation for 100% Architecture Compatibility, *24th Intl. Sym. on Comp. Architecture*, pp. 26-37, Jun 1997.

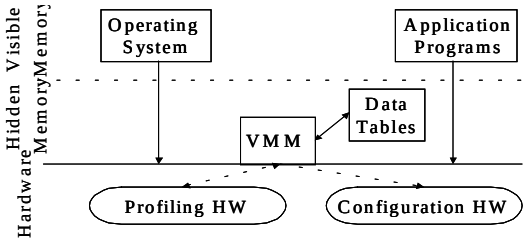

Figure 12.2.1: The co-designed virtual machine architecture. The physical main memory space addressable by virtual machine software is larger than the memory space addressable by software supported by the architected hardware. Code and data to be used by the VMM are placed in hidden memory and the VMM may have access to implementation-dependent instructions that interact with hardware performance monitoring and configuration

Benchmark	# Dynamic Phases	# Static Phases	95th %	Average Length	% time stable	% time unstable
applu	877	30	13	20	89	11
apsi	250	19	7	76	95	5
mgrid	2433	11	4	6	69	31
wupwise	26	6	5	767	100	0
perl	3746	59	17	2	45	55
gzip	2927	14	6	4	62	38

Figure 12.2.4: Statistics captured by the phase detection mechanism. The bit vector size is 4096 and the threshold used is .125. The columns from left to right show the benchmark name, number of dynamic phases detected, number of static phases detected, number of static phases that lead to 95% of the stable time, average stable phase length in units of 100K instructions, % of time in stable phases and % of time in unstable regions.

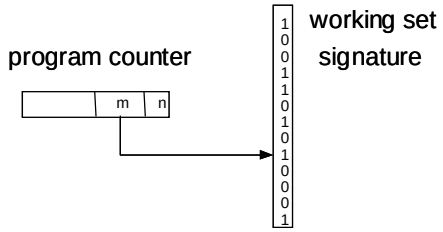

Figure 12.2.2: Mechanism for collecting working set signatures. m bits selected from the program counter are used to address a table containing 2^m bits. The table is cleared at the beginning of each sampling interval, and a bit is set if a corresponding branch instruction is executed.

```
if (state == STABLE)
    if (working_set_change)
        state = UNSTABLE;

else if (state == UNSTABLE)
    if (NOT working_set _change)
        do table lookup;
        if (entry found)
            unit_size = table_entry_size;
            state = STABLE;
        else
            unit_size = SMALLEST;
            state = TUNING;

else if (state == TUNING)
    if (NOT working_set _change)
        if (all combinations tried)
            select best configuration with performance
            within 2% of best performance;
            make table entry;
            state = STABLE;
        else
            try next configuration;
    else
        state = UNSTABLE;
```

Figure 12.2.3: Performance tuning in a VMM. This sequence is repeated after every execution interval (100K instructions).

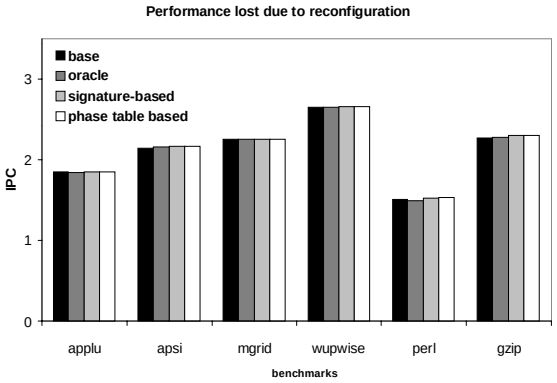

Figure 12.2.5: Performance as a result of reconfiguration. The base case has 32K I and D caches and a 4K entry predictor.

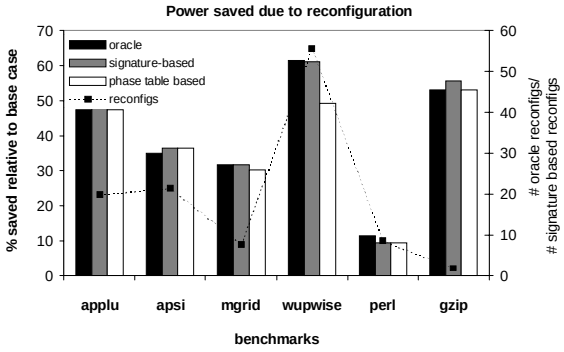

Figure 12.2.6: Power savings as a result of re-configuration. The base case has 32K I and D caches and a 4K entry predictor. The line shows the ratio of reconfigurations performed by the oracle algorithm to reconfigurations performed by the signature-based algorithm.