Parallel Architecture, Assignment 1

Mark Nemec, s1140740

1 Paper C: Superspeculative Microarchitecture for Beyond AD 2000

1.1 Description

This paper describes a class of techniques termed superspeculative and provides an implementation of these techniques in a microarchitecture called Superflow. It claims that this new paradigm gets performance increases over top-of-the-line processors of the time by predicting values produced by producer instructions and executing consumer instructions before the values become known.

The main reason to speculate about values of instructions is the classical dataflow limit for program performance:

> *Given unlimited machine resources, a program cannot execute any faster than the execution of the longest dependence chain induced by the program's true data dependences.*

This means that even with a very wide conventional superscalar the performance is limited by the serialisation of producer and consumer instructions. The only way to speed up the execution is to try to predict values before they become known. If the prediction is successful, this "breaks" the serialisation.

At the time, most machines adopted a strong-dependence model. This model specifies that instructions are possibly executed out-of-order but only if there is no dependence. What this means is that dependences between instructions are never violated. The authors claim that this model is "too rigorous and unnecessarily restricts available parallelism".

To be able to predict operands of consumer instructions, the authors propose a weak-dependence model. In this model the machine can temporarily violate dependences and speculate about operand values as long as it can recover from misspeculations. If a substantial number of predictions is correct, the paper claims that the machine can outperform, traditional strong-dependence machines.

1.2 Results

In Superflow, the authors implemented superspeculative techniques that improve instruction flow, register dataflow and memory dataflow. They claim that the techniques frequently more than double the performance of a conventional superscalar processor. In fact, Superflow has a potential performance of 9 instructions per cycle and simulations yielded 7.3 IPC for the SPECint95 benchmark suite. These results were possible without recompilation or changes to the instruction set architecture (ISA). A chart with performance comparison to a conventional superscalar can be observed in figure 1.

1.3 Discussion

I believe the main idea of the paper is still valid even in present day. For example, the instruction flow techniques presented in the paper such as branch predictor and trace cache have been incorporated into current-day processors and are still an active area of research. Furthermore, the technique called *value stride prediction* still has its use, while traversing an array, for example. I think where the paper fell short was not optimising for multi-threaded operating systems and programs which have become pervasive.

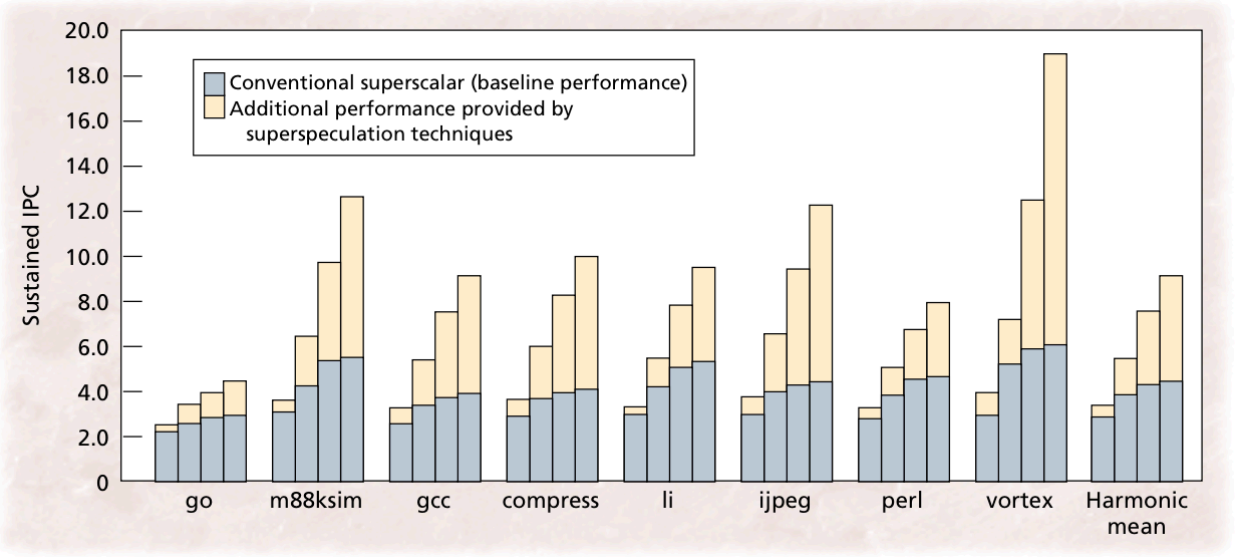

Figure 1: IPC of superscalar & superspeculative with 4, 8, 16, and 32 issue width

To detect dependencies between instructions every instruction needs to be cross-checked with every other instruction executing at the same time. This increases hardware complexity as well cycle time and even with the speculative techniques suggested in the paper one cannot hope to predict every dependency. As we can see in IPC of harmonic mean in figure 1, big performance difference came only with issue width of 16 and 32. For issue with of 4 and 8 the performance improvement was less than 2 IPC. Current desktop processors have issue width less than 8. Thus I believe this paper significantly overestimated the practicality of wide-issue superscalar processors.

The values of most instructions cannot be predicted (if they could be there would be no point in computing them). Thus even with a very large instruction window size, one could not hope to achieve massive performance improvements. Thus chip makers have turned to multiprocessors which are able to provide these improvements assuming that programmers write multi-threaded programs.

2 Paper B: A Single-Chip Multiprocessor

2.1 Description

This paper compares three different architectures that try to exploit parallelism available in computer programs: superscalar, simultaneous multi-threading (SMT) and chip multiprocessor (CMP). The superscalar architecture has one CPU and can execute 12 instructions in parallel. The SMT also has one CPU and same issue width but can run up to 8 threads. The CMP has 8 CPU cores, all on one chip, and each of these CPUs can run only one thread and has an issue width of 2.

2.2 Results

According to the results of simulations ran by the authors, the CMP architecture offered superior performance with simpler hardware than both SMT and superscalar architectures. A chart with relative performance can be observed in figure 2.

For code that can be parallelised into threads, the CMP architecture performs better or equally well as the more complicated wide-issue superscalars or SMT architectures. The SMT architecture has a more efficient resource utilisation than CMP but CMP can include more execution units in the same area due to its smaller issue width.

For code that cannot be parallelised into threads, the CMP architecture lags behind the other options due to its simpler design. However, the difference in performance is only minor because code that cannot be parallelised into threads usually cannot make much use of the wider issue-width of superscalar and SMT processors.

2.3 Discussion

One of the main reasons the CMP design succeeded is because of its simpler hardware. It is a composition of smaller, identical CPUs. Since these CPUs use ISA built on top of ISA of

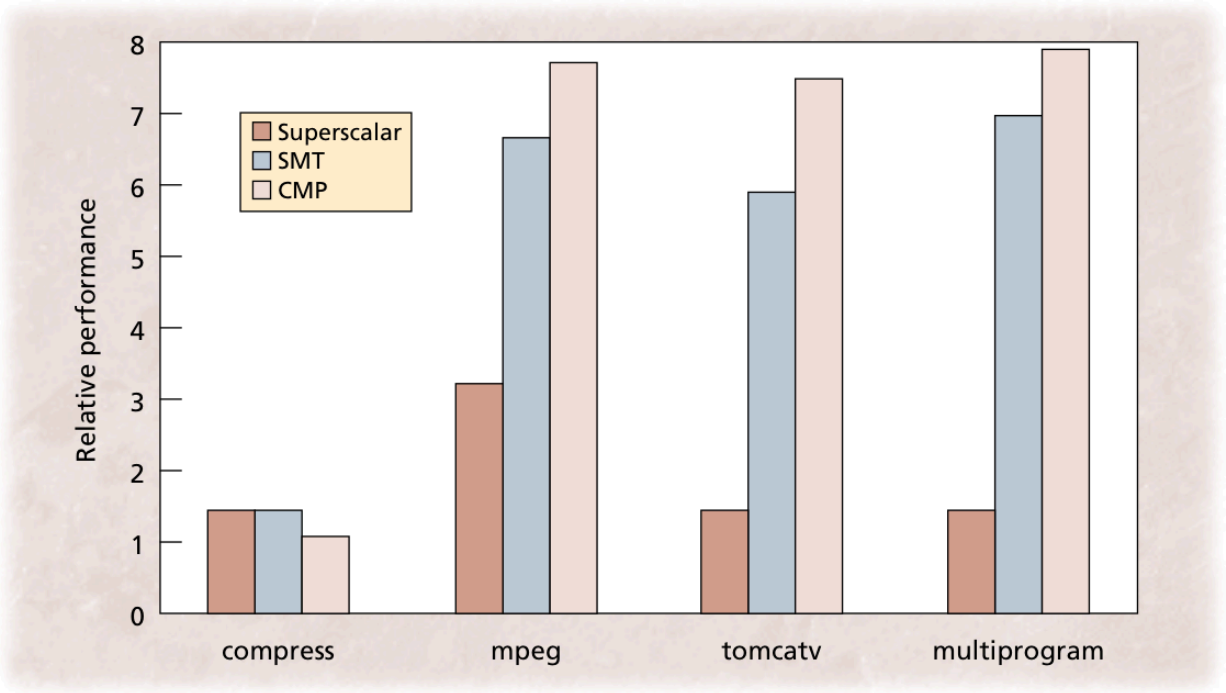

Figure 2: Relative performance of superscalar, SMT and CMP

single-core architectures of previous era there were no backward-compatibility issues.

The main factors driving commercial designs to adopt the single-chip multiprocessor approach were:

- **Good scaling:** The CMP architecture uses a group of small, identical CPUs. Since the processor is relatively simple the implementation costs are lower and since they are identical this cost is paid only once. Furthermore, since the CPUs are clustered, it scales well if the number of cores is ever increased. This is in contrast, for example, with superscalars where the number of registers and their ports must increase proportionally to the instruction window size.
- **Mapping threads to cores:** In the CMP architecture the operating system allocates a single thread to every processor. This allows the cores to be small and fast. Moreover, there does not need to be any hardware allocating instructions to different cores. This

hardware could severely lengthen the cycle time for an SMT architecture.

- **Compilers:** Exploiting instruction level parallelism in a compiler is difficult when your CPU can handle up to 12 instructions at a time. And where do you go once you reach the limit of instructions that can be executed in parallel? Clearly, the wide-issue processor solution has a limit. As the paper states, it is believed this is about 10-15 instructions per cycle. On the other hand, programmers could parallelise their programs by running multiple threads per application. The compiler then knows the threads can run in parallel and even if they need to communicate frequently the proximity of the cores on a chip ensures this communication is fast.

The most popular commercial processors of the last 10 years or so are all single-chip multiprocessors: Intel Core i Series, IBM Power8, and AMD Bulldozer. However, chip makers have lately been trying to combine CMP and SMT. An example of this is Intel's hyper-threading technology and AMD's clustered multi-thread (CMT) technology. This way the processors utilise resources that would otherwise be idle.