1 | A Single-Chip Multiprocessor - Paper B

1.1 | Description

The paper in question introduces two main concepts of tackling the problem of a large number of transistors - simultaneous multithreading (SMT) and a chip multiprocessor (CMP) and compares both to a superscalar processor. Additionally, the paper discusses concerns about the latency of interconnect delays, the need for parallelism and the impact of increased number of transistors on design complexity.

The SMT design proposed focuses on the use of multiple threads of execution each capable of issuing multiple instructions at once. It features dynamic selection of instructions to increase utilization. In the absence of thread level parallelism, the SMT appears as a conventional superscalar processor. Conversely, the CMP design focuses on utilizing multiple cores on a single die with the ability to issue multiple instructions at the same time, however, with no multithreading at the core level. The primary target of this design are environments where multi processes can be utilized and therefore allow each core handle the execution of an individual thread.

Additionally, the paper raises concerns about the increased delay of interconnect wires due to increased size. As a result, it proposes the use of partitioned sections of the architecture in order to keep interconnects short and simple. Furthermore, the paper emphasises the need for parallelism for future computing environments in order to provide a more sophisticated computing environment. Finally, the paper addresses concerns of design complexity and the space (area) requirements for the chip. In both instances, it is argued that increased complexity and computing performance will lead to disproportionately larger increase in the size of the chip. It is argued that the CMP solution is better suited to tackle both complexity and space challenges.

1.2 | Results

The results presented to support the arguments of the paper are obtained through a simulation of the hardware architecture in question based on a DRAM with 1 billion transistors. In order to establish a comparison, a superscalar, SMT and a CMP architectures are simulated. As a baseline “a single 2-issue processor attached to the superscalar/SMT memory system” [1] is used.

The specification of the architectures varies mainly in their design. In the case of the SMT and CMP, resources are partitioned into their respective threads or cores respectively, with the exception of issue width where the CMP has an issue width of 2 for each CPU, with eight CPUs, this leads to a total issue width of 16. The superscalar and the SMT both have issue width of 12. The SMT and superscalar architectures feature advanced branch prediction.

On a non-thread level parallelism benchmark, the superscalar and SMT both perform 43% better than the baseline while the CMP performs similarly to the baseline. This is due to the ability to utilize the larger issue width. On the other hand, CMP is not able to obtain a speedup due to the non-threaded nature of the application.

A large thread-level parallelism benchmark reveals both SMT and CMP performing very well, achieving close to 7 and 8 times speedup respectively. The superscalar benchmark performs well, however, only achieving about 3.5 speedup. CMP outperforms SMT due to larger number of issue slots. The superscalar processor is unable to uncover thread level parallelism in the program and lacks behind.

The remaining benchmarks focusing on both thread-level and instruction-level parallelism and multiprogramming scenario both show similar results. The CMP performs the best followed by SMT. The results obtained are similar to the scenario with large thread-level parallelism discussed previously.

1.3 | Discussion

Firstly, the proposed challenge of increased delay of interconnects used in CPUs has persisted, however, it has been addressed through the use of improved semiconductor device fabrication allowing the length of interconnects to be decreased and therefore improve latency.

Secondly, the concept of the chip multiprocessor has been adopted and has become a standard way to increase performance of a chip. Due to the power wall, it has become infeasible to keep increasing the clock rate of CPUs and instead multiple CPUs are now deployed on a single chip.

Thirdly, the idea of large SMT CPUs has not become prevalent. Instead, moderate levels of simultaneous multithreading coupled with multiprocessor design has been adopted. This is due to inherent patterns of execution where insufficient thread level parallelism and data locality prevents a large number of threads of execution to be used effectively. Additionally, as memory performance lacked behind CPU performance, it became increasingly difficult to provide sufficient latency for large SMT execution model.

Furthermore, the idea of clustering of resources together has been adopted. For example, in modern chips level 1 caches are co-located with their respective cores. Similarly, the instruction caches are dedicated to individual cores to improve instruction issue.

Moreover, operating systems and applications have become significantly better at resource allocation and the use of multi-processor architectures. Additionally, application development tools have improved allowing for application design to exploit both threading and multiprocessing significantly.

Finally, majority of the ideas proposed by the paper have been implemented to some degree. A combination of the approaches outlined has been adopted to target instruction level parallelism, thread level parallelism and process level parallelism.

2 | One Billion Transistors, One Uniprocessor, One Chip - Paper C

2.1 | Description

The study argues that a single chip will be able to deliver the best performance in the one billion transistor age. It proposes optimized components such as a trace cache, branch prediction and data supply improvements to support the design.

Firstly, it is argued it is essential to issue the maximum number of instructions as possible. Subsequently, it is also essential to provision sufficient bandwidth to consume the results of the instructions.

Secondly, the concept of a trace cache is presented. A trace cache stores a sequence of instruction executed in logical order rather than in issue order in order to avoid stalls due to branching.

Thirdly, the requirement for advanced branch prediction is proposed. An implementation similar to MultiHybrid is suggested as well as the need to be able to predict context switching, jumps and interference handling. It is proposed the compiler aids the architecture with branch prediction.

Furthermore, the issue of memory and data supply is presented. In order to limit stalls, out of order issue is suggested. Additionally, a replicated first level cache is introduced to alleviate access contention.

Finally, the need for register renaming and a large number of functional units is presented. It is argued that communication delay consisting of routing, scheduling and data forwarding will be the largest source of delay and clustering of functional units is proposed as a solution.

2.2 | Results

Firstly, benchmark results of a proposed trace cache are presented. The benchmark focuses on the number of instructions issued per cycle compared to the size of the cache. Traditionally, an increase in instruction cache size suffers from diminishing returns, however, the trace cache scales better with cache size. The benchmark compares the two caches on three different datasets. On average, the trace cache allows 2 to 3 more instructions to be issued, peaking at 6 to 7 instructions per cycle while the standard instruction cache only achieves 4 to 5 instructions per cycle.

Secondly, misprediction benchmark is examined for the MultiHybrid branch prediction model. The benchmark focuses on misprediction rate vs the size of the branch predictor. The misprediction rate decreases linearly with the size of the branch predictor, starting at 5% at 16 Kbytes and decreasing to 3% at 256 Kbytes.

Finally, a relationship between issue/execution window and the number of instructions per cycle is examine. The results show “wider issue and execution widths allow the parallelism within the program to be exploited” [2]. This is under the assumption of perfect branch prediction which remains impossible today.

2.3 | Discussion

A subset of the design concepts proposed have been utilized in practice, however, some ideas have been used in limited scope. In particular, the concept of a single processor has been superseded by multi-processing. Therefore, the main proposed design of the paper has not stood the test of time while individual suggestions on how to achieve the uniprocessor design have been exploited and introduced into the multi-processing paradigm.

Firstly, the proposed trace cache ended up being implemented in modern CPUs. It lead to an in increase in execution time of the CPU by storing already decoded instructions in the cache and accessing the logical sequence of instructions. The Intel Pentium 4 architecture used.

Secondly, advanced branch prediction has become a widely used strategy to avoid stalls. The proposed methods of branch prediction have indeed become utilized, however, additional methods such as neural network branch predictions have been added to the general branch prediction at a later stage.

Thirdly, out of order issue proposed has also been deployed. The model is governed by availability of data rather than the logical order of instructions.

On the other hand, the concept of replicated first level cache has been superseded by utilizing a dedicated first level cache with interconnects designed to communicate shared variable changes.

Furthermore, the proposal to design the architecture to support wide-issue has not been implemented in its proposed form. Fundamental data hazard and an inherent degree of unpredictability of branching led to the inability to fully exploit wide issue at the levels proposed. Wide-issue is, however, used in practice albeit in smaller form.

Ultimately, the idea of “prefetch and predict” predict becoming the norm of modern architecture has persisted. The paper has failed to predict the direction, however, in doing so, the paper succeeded in suggesting methods which in fact turned out to be very successful.

References:
[1] http://www.inf.ed.ac.uk/teaching/courses/pa/Papers/billion_chipmultiprocessor.pdf
[2] http://www.inf.ed.ac.uk/teaching/courses/pa/Papers/billion_uniprocessor.pdf