# Parallel Processing

Anirudh Krishna Villivalam,
Jennifer Xiao

# Agenda

Multiscalar Processors

The Case for a Single-Chip Multiprocessor

Paper Discussion

# Multiscalar Processors

# Motivation

- Long history (50 years) of sequential coding lead to a style of writing code assuming instructions execute in the order in which they are written.
- This changed with the introduction of processors that are able to perform out-of-order parallel execution (ILP).
- But out-of-order execution has few hazards, such as data and control, that can substantially slow the parallel execution.
- A control flow graph (CFG) can be used to tackle control dependencies.
- The paper focuses on a multiscalar approach with a CFG that can be used to exploit fine-grain or instruction level parallelism.

# Main Contribution

- Describes a new multiscalar paradigm with the use of CFG.
- Provides insight on how to efficiently distribute processing unit cycles.
- Challenges the conventions regarding ILP.

# Technical Assumptions

- Overhead involved in task synchronization is minimal.
- Sequencer does a good job identifying and assigning tasks.
- Tasks are either completely executed or squashed.

# Merits

- Multiscalar processors can handle control dependencies efficiently.
- Useful for cases where dependency between instructions can not be determined before program execution.
- Provides high branch prediction across multiple branches.
- Reduces complexity for monitoring instructions.
- Reduces logic complexity required for n instructions.
- Allows loads and stores to be issued independently within one task.
- Uses both hardware and software for re-ordering instructions.

# Failings

- High IPC because of aggressive processing units.
- Increased latency for cache hits.
- Additional instructions are required for multiscalar execution.
- Requires additional hardware as well.

# Methodology

- Concept of Control Flow Graph and Multiscalar architecture is introduced.
- The multiscalar model uses partitions called tasks which are assigned to the processing units.
- Tasks are defined as part of CFG that corresponds to some contiguous region from the set of instructions.
- A microarchitecture is described with an example CFG.
- The distribution of available cycles is analyzed.
- A comparison of the multiscalar architecture with other paradigms is provided.
- The performance of the architecture is compared with other paradigms such as scalar, VLIW, superscalar and multiprocessors.
- Lastly, the performance of the architecture with respect to scalar architecture is presented.

# Overview of the paradigm presented

- The purpose of the CFG is to ensure a large and accurate window from which instructions can be extracted and scheduled dynamically.
- A task is some part of this CFG which is assigned to a processing unit.
- All the instructions in each task are bounded by the first and last instruction in that task.
- Each processing unit executes instructions of the task.
- Tasks need not be independent of each other. To ensure communication among the tasks a unidirectional ring can be used.
- For maintaining overall sequential appearance, each processing unit executes instructions of the task sequentially.
- Additionally, the processing units themselves follow a loose sequential order

# One possible architecture

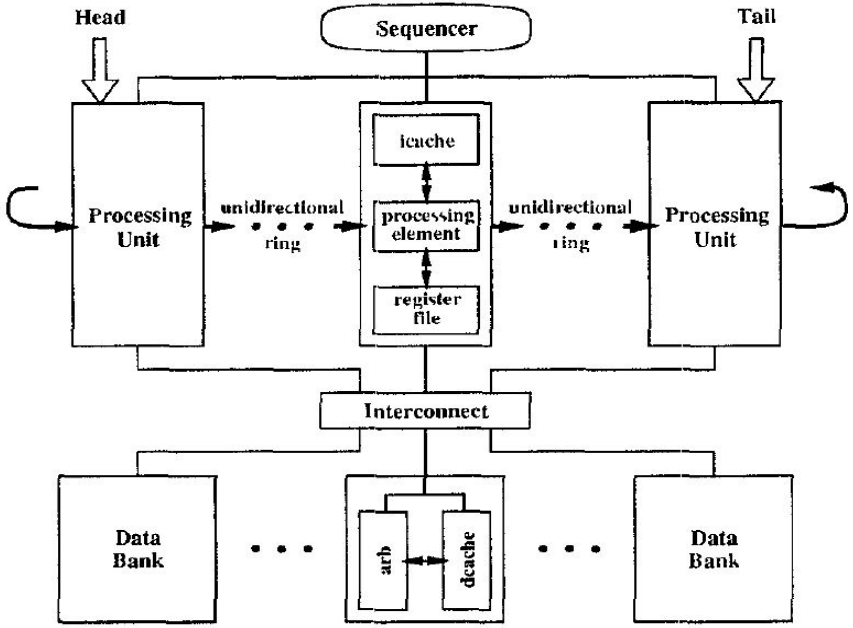


**Figure 1:** A Possible Microarchitecture of a Multiscalar Processor.

# Writing multiscalar programs

- The multiscalar program needs to ensure there is sufficient support for using the CFG
- The sequencer needs the information regarding program flow to enable prediction of the next task to be assigned. This allows for the next task to be assigned without spending time on inspecting the instructions of the present task.
- Additional tag bits are required for stopping and forwarding instructions.
- Writing multiscalar programs from existing code is possible by adding the required tag and task descriptor bits. This allows for some portability from one generation of hardware to another.

# Distributing cycles


- The aim of the multiscalar approach is to ensure each processing unit executes multiple instructions in a given cycle
- Cycles in which the unit does not perform useful computation, performs no computation or remains idle causes the performance to drop from the best case.
- Non-useful computation occurs when a task needs to be squashed. This can happen due to either an incorrect value or an incorrect prediction.
- Synchronizing data communication and performing early prediction can help prevent some non-useful computation.

# Distributing cycles

- Managing intra-task dependencies by using code scheduling, non-blocking cache and out-of-order execution can reduce no computation cycles.
- Inter-task dependencies are more prevalent in the multiscalar approach. This can somewhat be dealt with by using early data updation and forwarding.
- Ensuring each task has approximately the same size is useful in minimizing lost cycles.

# Evaluation

- The multiscalar processor simulator used performed all the tasks and operations with the exception of system calls.
- A 5 stage pipeline structure was used with the options to configure it as in order/ out of order and 1-way or 2-way issue.
- Ten programs were used with some of them having modifications. Almost all of them have a significant number of loops. Perhaps this was on purpose to highlight the aggressive parallel execution provided by the multiscalar approach.

# Conclusions

| Program | Instruction Count | | Percent Increase |
|---|---|---|---|
| | Scalar | Multiscalar | |
| Compress | 71.04M | 81.21M | 14.3% |
| Eqntott | 1077.50M | 1237.73M | 14.9% |
| Espresso | 526.50M | 615.95M | 17.0% |
| Gcc | 66.48M | 75.31M | 13.3% |
| Sc | 409.06M | 460.79M | 12.6% |
| Xlisp | 46.61M | 54.34M | 16.6% |
| Tomcatv | 582.22M | 590.66M | 1.4% |
| Cmp | 0.98M | 1.09M | 10.9% |
| Wc | 1.22M | 1.43M | 17.3% |
| Example | 1.05M | 1.09M | 4.2% |

**Table 2:** Benchmark Instruction Counts.

# Conclusions

| Program | 1-Way Issue Units | | | | | 2-Way Issue Units | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Scalar IPC | Multiscalar | | | | Scalar IPC | Multiscalar | | | |
| | | 4-Unit | | 8-Unit | | | 4-Unit | | 8-Unit | |
| | | Speedup | Pred | Speedup | Pred | | Speedup | Pred | Speedup | Pred |
| Compress | 0.69 | 1.17 | 86.8% | 1.50 | 86.1% | 0.87 | 1.04 | 86.8% | 1.34 | 86.4% |
| Eqntott | 0.83 | 2.05 | 94.8% | 2.91 | 94.6% | 1.10 | 1.82 | 94.8% | 2.58 | 94.6% |
| Espresso | 0.85 | 1.34 | 85.9% | 1.59 | 85.9% | 1.11 | 1.22 | 85.3% | 1.41 | 85.2% |
| Gcc | 0.81 | 1.02 | 81.2% | 1.08 | 80.9% | 1.04 | 0.92 | 81.2% | 0.98 | 80.9% |
| Sc | 0.75 | 1.36 | 90.5% | 1.68 | 90.0% | 0.94 | 1.28 | 90.0% | 1.56 | 89.5% |
| Xlisp | 0.80 | 0.91 | 80.6% | 0.94 | 79.5% | 1.03 | 0.86 | 80.0% | 0.88 | 78.7% |
| Tomcatv | 0.80 | 3.00 | 99.2% | 4.65 | 99.2% | 0.97 | 2.71 | 99.2% | 3.96 | 99.2% |
| Cmp | 0.95 | 3.23 | 99.4% | 6.24 | 99.4% | 1.32 | 3.02 | 99.4% | 5.82 | 99.4% |
| Wc | 0.89 | 2.37 | 99.9% | 4.33 | 99.9% | 1.09 | 2.36 | 99.9% | 4.27 | 99.9% |
| Example | 0.79 | 2.79 | 99.9% | 3.96 | 99.9% | 1.07 | 2.43 | 99.9% | 3.47 | 99.9% |

**Table 3:** In-Order Issue Processing Units.

| Program | 1-Way Issue Units | | | | | 2-Way Issue Units | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Scalar IPC | Multiscalar | | | | Scalar IPC | Multiscalar | | | |
| | | 4-Unit | | 8-Unit | | | 4-Unit | | 8-Unit | |
| | | Speedup | Pred | Speedup | Pred | | Speedup | Pred | Speedup | Pred |
| Compress | 0.72 | 1.23 | 86.7% | 1.56 | 86.0% | 0.94 | 1.07 | 86.7% | 1.33 | 86.3% |
| Eqntott | 0.84 | 2.23 | 94.8% | 3.35 | 94.6% | 1.21 | 1.79 | 94.8% | 2.64 | 94.5% |
| Espresso | 0.88 | 1.47 | 85.9% | 1.73 | 85.8% | 1.31 | 1.12 | 85.3% | 1.25 | 85.4% |
| Gcc | 0.83 | 1.06 | 81.1% | 1.13 | 80.6% | 1.15 | 0.91 | 81.1% | 0.95 | 80.6% |
| Sc | 0.80 | 1.42 | 90.5% | 1.75 | 90.0% | 1.10 | 1.24 | 90.2% | 1.50 | 90.2% |
| Xlisp | 0.82 | 0.95 | 75.6% | 1.01 | 77.1% | 1.12 | 0.85 | 74.6% | 0.90 | 76.5% |
| Tomcatv | 0.96 | 2.92 | 99.2% | 4.17 | 99.2% | 1.43 | 2.16 | 99.2% | 2.93 | 99.2% |
| Cmp | 0.95 | 3.24 | 99.2% | 6.28 | 99.1% | 1.68 | 2.76 | 99.2% | 5.30 | 99.2% |
| Wc | 0.89 | 2.37 | 99.9% | 4.34 | 99.9% | 1.13 | 2.34 | 99.9% | 4.26 | 99.9% |
| Example | 0.86 | 3.27 | 99.9% | 4.86 | 99.9% | 1.28 | 2.41 | 99.9% | 3.57 | 99.9% |

**Table 4:** Out-Of-Order Issue Processing Units.

# The Case for a Single-Chip Multiprocessor

# Motivation

- Diminishing returns on making superscalar processors wider
  - Wider superscalar processors require quadratically more logic and wires, limiting frequency and increasing power
  - Performance is only fractionally better for processors twice as wide
- Single-Chip Multiprocessors allow for better extraction of parallelism by software developers, and better performance per chip area

# Main Contribution

- Change in thought process about how to go about creating processors
  - One very wide superscalar processor or single-chip multiprocessor?
- Proposed an area efficient alternative to the single superscalar processor
- The single-chip multiprocessor architecture allows for fine-grained parallelism extraction by software developers / multithreaded software

# Technical Assumptions

- IPC numbers are not actually given for multiprocessor results - only cache miss rates -- this somehow translates to speedup
- They assume they can directly compare the architectures when the microarchitectures that their architectures are based on are different.
- Assumed that a 6-way architecture, which the simulation code is not optimized for, was comparable to 4 2-way processors.

# Merits

- Single-chip multiprocessor doesn’t imply not using superscalar processors
  - Retain the best of both architectures
- Extracts coarse-grained parallelism better than superscalar processors
- Power efficiency of multiple smaller cores became important when we hit the power wall

# Failings

- Nonzero thread synchronization cost for multithreaded applications
- Purely sequential applications do not benefit from multiple cores, and perform better on larger superscalar cores
- Puts more of the burden of performance on software developers

# Methodology

- Authors developed two microarchitectures for hypothetical machines in the future
  - “Logical extension” of the current 4-way superscalar R10000 superscalar design into a 6-way superscalar design
    - Additionally increased size of instruction buffers / instruction window
  - Multiprocessor architecture: 4-way single chip multiprocessor with 4 2-way superscalar processors. Each is ~= the Alpha 21064
- Authors then simulated nine applications in the SimOS environment, measuring performance in the representative execution window
    - SPEC95 compress and m88ksim, SPEC92 eqntott, MPsim, SPEC95 applu

# Methodology

- Authors then simulated nine applications in the SimOS environment, measuring performance in the representative execution window using the most detailed simulator (MXS), and less detailed but faster simulators for the rest
    - Integer benchmarks: SPEC95 compress and m88ksim, SPEC92 eqntott, MPsim
    - FP benchmarks: SPEC95 applu, apsi, swim, and tomcatv
    - Multiprogramming benchmark: pmake (measured fully in MVS due to lack of clear representative window)

# Proposed Floorplans

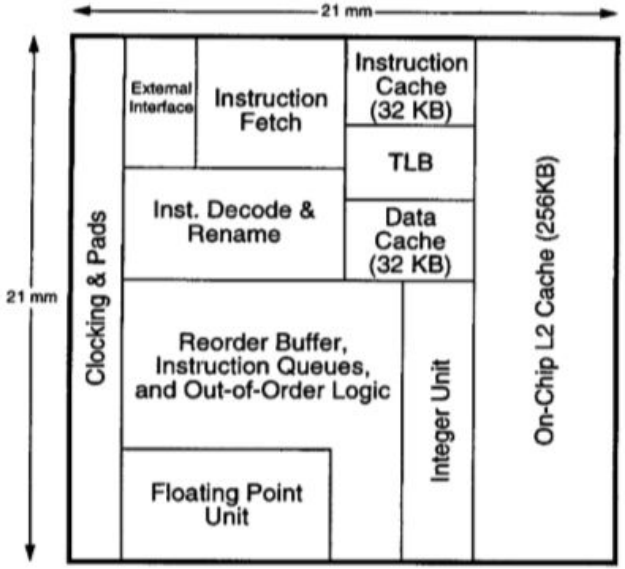


**Figure 2. Floorplan for the six-issue dynamic superscalar microprocessor.**

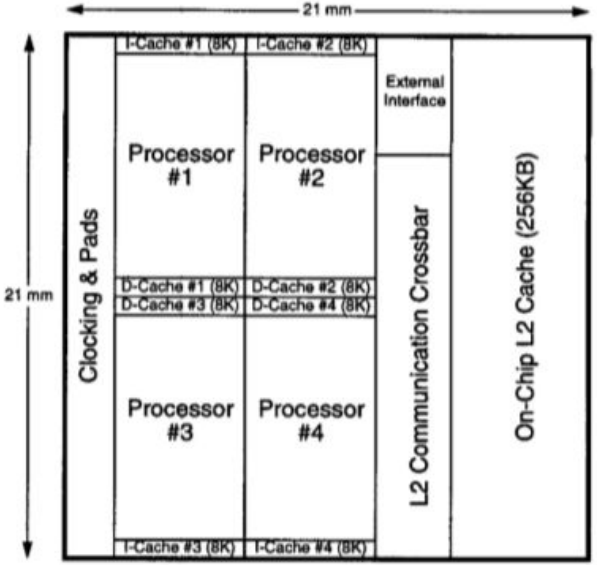


**Figure 3. Floorplan for the four-way single-chip multiprocessor.**

# Proposed Characteristics

| | 6-way SS | 4x2-way MP |
|---|---|---|
| # of CPUs | 1 | 4 |
| Degree superscalar | 6 | 4 x 2 |
| # of architectural registers | 32int / 32fp | 4 x 32int / 32fp |
| # of physical registers | 160int / 160fp | 4 x 40int / 40fp |
| # of integer functional units | 3 | 4 x 1 |
| # of floating pt. functional units | 3 | 4 x 1 |
| # of load/store ports | 8 (one per bank) | 4 x 1 |
| BTB size | 2048 entries | 4 x 512 entries |
| Return stack size | 32 entries | 4 x 8 entries |
| Instruction issue queue size | 128 entries | 4 x 8 entries |
| I cache | 32 KB, 2-way S. A. | 4 x 8 KB, 2-way S. A. |
| D cache | 32 KB, 2-way S. A. | 4 x 8 KB, 2-way S. A. |
| L1 hit time | 2 cycles (4 ns) | 1 cycle (2 ns) |
| L1 cache interleaving | 8 banks | N/A |
| Unified L2 cache | 256 KB, 2-way S. A. | 256 KB, 2-way S. A. |
| L2 hit time / L1 penalty | 4 cycles (8 ns) | 5 cycles (10 ns) |
| Memory latency / L2 penalty | 50 cycles (100 ns) | 50 cycles (100 ns) |

**Table 1. Key characteristics of the two microarchitectures**

# Conclusions

| Program | IPC | BP Rate % | I cache %MPCI | D cache %MPCI | L2 cache %MPCI |
|---|---|---|---|---|---|
| compress | 0.9 | 85.9 | 0.0 | 3.5 | 1.0 |
| eqntott | 1.3 | 79.8 | 0.0 | 0.8 | 0.7 |
| m88ksim | 1.4 | 91.7 | 2.2 | 0.4 | 0.0 |
| MPsim | 0.8 | 78.7 | 5.1 | 2.3 | 2.3 |
| applu | 0.9 | 79.2 | 0.0 | 2.0 | 1.7 |
| apsi | 0.6 | 95.1 | 1.0 | 4.1 | 2.1 |
| swim | 0.9 | 99.7 | 0.0 | 1.2 | 1.2 |
| tomcatv | 0.8 | 99.6 | 0.0 | 7.7 | 2.2 |
| pmake | 1.0 | 86.2 | 2.3 | 2.1 | 0.4 |

**Table 5. Performance of a single 2-issue superscalar processor.**

| Program | IPC | BP Rate % | I cache %MPCI | D cache %MPCI | L2 cache %MPCI |
|---|---|---|---|---|---|
| compress | 1.2 | 86.4 | 0.0 | 3.9 | 1.1 |
| eqntott | 1.8 | 80.0 | 0.0 | 1.1 | 1.1 |
| m88ksim | 2.3 | 92.6 | 0.1 | 0.0 | 0.0 |
| MPsim | 1.2 | 81.6 | 3.4 | 1.7 | 2.3 |
| applu | 1.7 | 79.7 | 0.0 | 2.8 | 2.8 |
| apsi | 1.2 | 95.6 | 0.2 | 3.1 | 2.6 |
| swim | 2.2 | 99.8 | 0.0 | 2.3 | 2.5 |
| tomcatv | 1.3 | 99.7 | 0.0 | 4.2 | 4.3 |
| pmake | 1.4 | 82.7 | 0.7 | 1.0 | 0.6 |

**Table 6. Performance of the 6-issue superscalar processor.**

| Application | I cache %MPCI | D cache %MPCI | L2 cache %MPCI |
|---|---|---|---|
| compress | 0.0 | 3.5 | 1.0 |
| eqntott | 0.6 | 5.4 | 1.2 |
| m88ksim | 2.3 | 3.3 | 0.0 |
| MPsim | 4.8 | 2.5 | 3.4 |
| applu | 0.0 | 2.1 | 1.8 |
| apsi | 2.7 | 6.9 | 2.0 |
| swim | 0.0 | 1.2 | 1.5 |
| tomcatv | 0.0 | 7.8 | 2.5 |
| pmake | 2.4 | 4.6 | 0.7 |

**Table 7. Performance of the 4 × 2-issue processor.**

# Conclusions

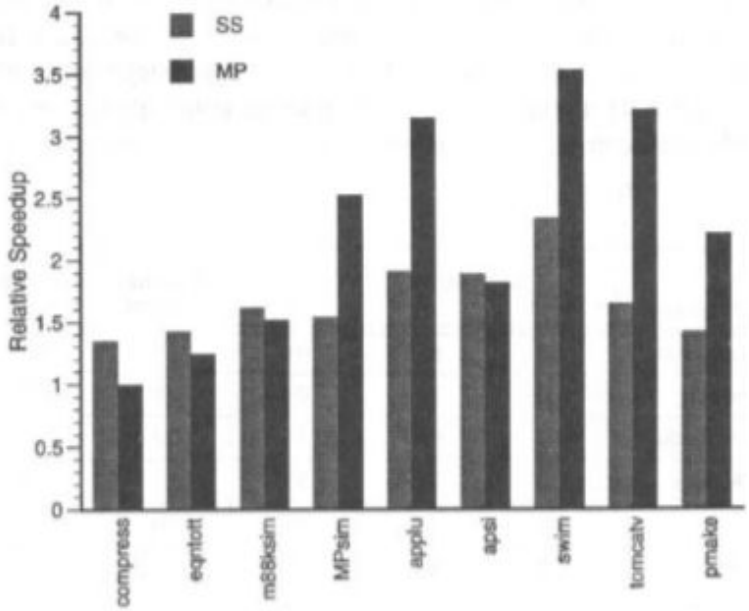


**Figure 6. Performance comparison of SS and MP.**

# Discussion Questions

# How relevant are these papers now?

# How realistic is a task-based multiscalar processor?

# Would an aggressively speculative multiscalar processor be insecure / vulnerable to Spectre/Meltdown?

# How do you think the single-chip multiprocessor author feels about GPUs?

# How do you think the single-chip multiprocessor author feels about modern CPUs?