Forming Virtual Traces for WCET Analysis and Reduction

Jack Whitham and Neil Audsley

August 27th 2008

Goal of this work

Propose CPU modifications for:

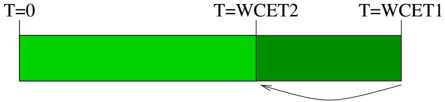
1. accurate *worst case execution time* (WCET) analysis.

Goal of this work

Propose CPU modifications for:

1. accurate *worst case execution time* (WCET) analysis.

2. improved *guaranteed throughput* (versus a simple CPU).



Requirements

The CPU modifications must:

- accommodate speculative and superscalar out-of-order operation so that throughput can be increased versus a simple CPU, and

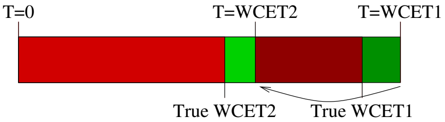
Requirements

The CPU modifications must:

- accommodate speculative and superscalar out-of-order operation so that throughput can be increased versus a simple CPU, and
- restrict this operation so that:
  - timings can be determined safely by measurement, and

The CPU modifications must:

- accommodate speculative and superscalar out-of-order operation so that throughput can be increased versus a simple CPU, and
- restrict this operation so that:
  - timings can be determined safely by measurement, and
  - the WCET analysis model won't include any *pessimistic* assumptions.



A trace

A trace is a path through a program.

A trace

A trace is a path through a program.

In this work, every trace has two forms:

1. *executable code* that implements some machine code within a program; often more than one *basic block*.
2. a *timing model* that gives precise information about *path timings* through that code.

A trace

A trace is a path through a program.

In this work, every trace has two forms:

1. *executable code* that implements some machine code within a program; often more than one *basic block*.
2. a *timing model* that gives precise information about *path timings* through that code.

Usually, the main path through a trace executes more quickly than the equivalent machine code. *Always*, the function of a program is unchanged by adding traces.

A trace

A trace is a path through a program.

In this work, every trace has two forms:

1. *executable code* that implements some machine code within a program; often more than one *basic block*.
2. a *timing model* that gives precise information about *path timings* through that code.

Usually, the main path through a trace executes more quickly than the equivalent machine code. *Always*, the function of a program is unchanged by adding traces.

How do traces meet the requirements?

Previous work

In previous work, we considered the use of a *trace scratchpad* to implement traces and meet the requirements, used as follows:

1. Take a program in machine code form;

Previous work

In previous work, we considered the use of a *trace scratchpad* to implement traces and meet the requirements, used as follows:

1. Take a program in machine code form;
2. Apply WCET analysis to find the WCEP;

Previous work

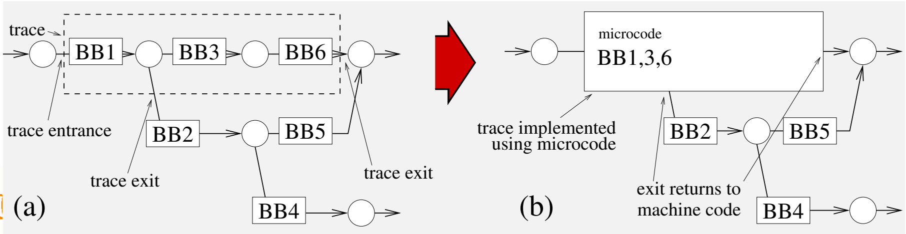
In previous work, we considered the use of a *trace scratchpad* to implement traces and meet the requirements, used as follows:

1. Take a program in machine code form;
2. Apply WCET analysis to find the WCEP;
3. Convert subsequences of the WCEP into *traces* implemented by microcode. These are *explicitly parallel* and *optimise execution for one path*.

Previous work

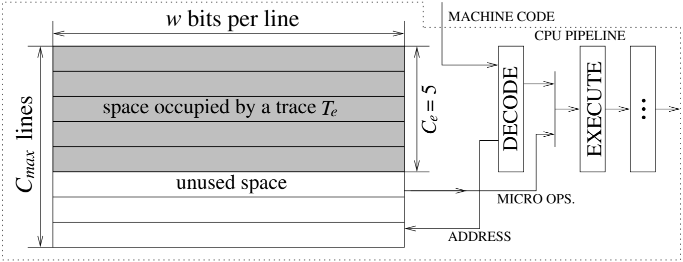
In previous work, we considered the use of a *trace scratchpad* to implement traces and meet the requirements, used as follows:

1. Take a program in machine code form;
2. Apply WCET analysis to find the WCEP;
3. Convert subsequences of the WCEP into *traces* implemented by microcode. These are *explicitly parallel* and *optimise execution for one path*.



Previous work

4. Allocate space in a *trace scratchpad* for microcode.
   The microcode is used in place of the original machine code.



J. Whitham and N. Audsley, Using trace scratchpads to reduce execution times in predictable real-time architectures, Proc. RTAS, 305–316, 2008.

Virtual trace

A virtual trace is a compact encoding, specifying the execution path that should be assumed by the CPU.

Virtual trace

A virtual trace is a compact encoding, specifying the execution path that should be assumed by the CPU.

Virtual traces are theoretically equivalent to traces, but some practical problems are solved:

- the need for a custom CPU with a writable microcode store,
- the need for a CPU-specific compiler to generate microcode,
- the poor code density of microcode.

Virtual trace

A virtual trace is a compact encoding, specifying the execution path that should be assumed by the CPU.

Virtual traces are theoretically equivalent to traces, but some practical problems are solved:

- the need for a custom CPU with a writable microcode store,
- the need for a CPU-specific compiler to generate microcode,
- the poor code density of microcode.

Result: No microcode. The virtual trace controls a conventional *but constrained* dynamic CPU scheduler.

Implementation

1. Regard the CPU dynamic scheduler as a decoder:

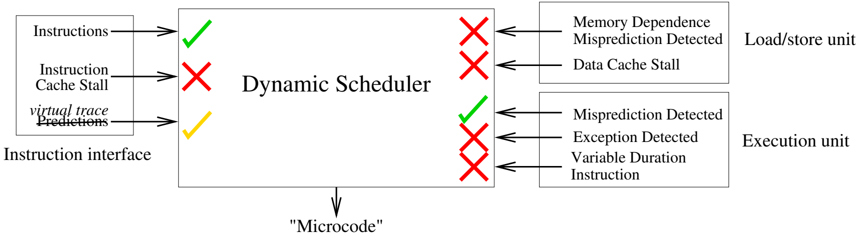
$$\textit{machine code} + \textit{virtual trace} \rightarrow \textit{microcode}$$

Implementation

1. Regard the CPU dynamic scheduler as a decoder:

$$\textit{machine code} + \textit{virtual trace} \rightarrow \textit{microcode}$$

2. Handle all events that could change execution times.



Benefits of virtual traces

Some are the same as for traces:

- Use speculative and superscalar out-of-order execution predictably.

Benefits of virtual traces

Some are the same as for traces:

- Use speculative and superscalar out-of-order execution predictably.
- The execution time of any path through any program is known.

Benefits of virtual traces

Some are the same as for traces:

- Use speculative and superscalar out-of-order execution predictably.
- The execution time of any path through any program is known.
- Use {*your favorite static WCET approach*} for analysis.

Benefits of virtual traces

Some are the same as for traces:

- Use speculative and superscalar out-of-order execution predictably.
- The execution time of any path through any program is known.
- Use {*your favorite static WCET approach*} for analysis.

But there's more:

- Any CPU could be modified with the correct restrictions. Predictable mode could be optional.

Benefits of virtual traces

Some are the same as for traces:

- Use speculative and superscalar out-of-order execution predictably.
- The execution time of any path through any program is known.
- Use {*your favorite static WCET approach*} for analysis.

But there's more:

- Any CPU could be modified with the correct restrictions. Predictable mode could be optional.
- The CPU is its own timing model.

Problem (for this paper)

How do we turn a program (as a graph of basic blocks) into a graph of virtual traces traces?

Problem (for this paper)

How do we turn a program (as a graph of basic blocks) into a graph of virtual traces traces?

- **Previously**: due to limited trace scratchpad space, only *some* parts of the program could be translated. A specialized search algorithm was used to find the most suitable WCEP subsequences.

Problem (for this paper)

How do we turn a program (as a graph of basic blocks) into a graph of virtual traces traces?

- **Previously**: due to limited trace scratchpad space, only *some* parts of the program could be translated. A specialized search algorithm was used to find the most suitable WCEP subsequences.
- **Now**: space limit is less restrictive, so the whole program should be translated.

Problem (for this paper)

How do we turn a program (as a graph of basic blocks) into a graph of virtual traces traces?

- **Previously**: due to limited trace scratchpad space, only *some* parts of the program could be translated. A specialized search algorithm was used to find the most suitable WCEP subsequences.
- **Now**: space limit is less restrictive, so the whole program should be translated.
- This problem is similar to selecting static branch predictions to minimize WCET.

F. Bodin and I. Puaut. A WCET-oriented static branch prediction scheme for real time systems. In Proc. ECRTS, pages 33–40, 2005.

Proposed solution

Combine two algorithms:

- Bodin-Puaut static branch prediction scheme.
- Trace formation algorithm from previous work.

Proposed solution

Combine two algorithms:

- Bodin-Puaut static branch prediction scheme.
    1. Assume all branches are *unknown*;
- Trace formation algorithm from previous work.

Proposed solution

Combine two algorithms:

- Bodin-Puaut static branch prediction scheme.
  1. Assume all branches are *unknown*;
  2. Use WCET analysis to find the WCEP through the program;
- Trace formation algorithm from previous work.

Proposed solution

Combine two algorithms:

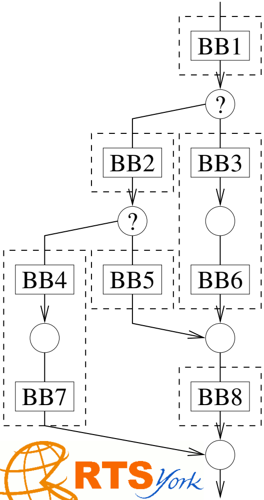
- Bodin-Puaut static branch prediction scheme.
  1. Assume all branches are *unknown*;
  2. Use WCET analysis to find the WCEP through the program;
  3. Assign *unknown* branches to follow the WCEP.
     If any *unknown* branches were found, go to 2.
- Trace formation algorithm from previous work.

Proposed solution

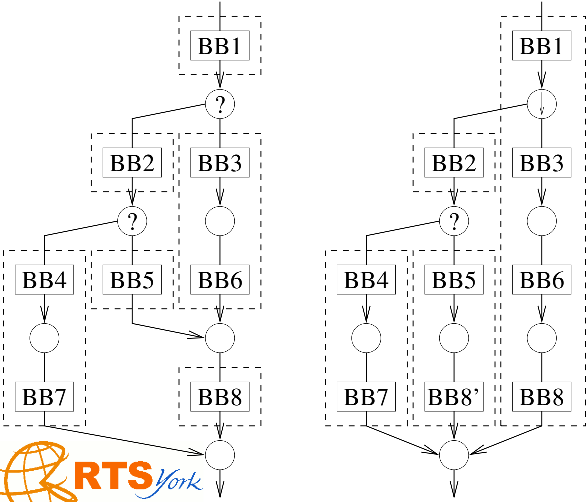
Combine two algorithms:

- Bodin-Puaut static branch prediction scheme.
  1. Assume all branches are *unknown*;
  2. Use WCET analysis to find the WCEP through the program;
  3. Assign *unknown* branches to follow the WCEP.
     If any *unknown* branches were found, go to 2.
- Trace formation algorithm from previous work.
  - Form traces by following branch predictions.

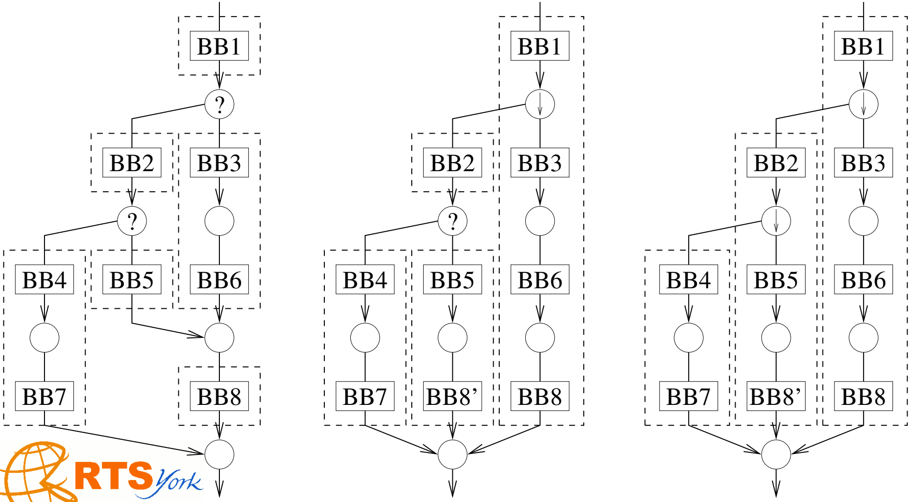
Example



Example


BB1
?
BB2
BB3
?
BB4
BB5
BB6
BB7
BB8
BB1
BB2
BB3
?
BB4
BB5
BB6
BB7
BB8'
BB8

Example



BB1
?
BB2
BB3
?
BB4
BB5
BB6
BB7
BB8
BB1
BB2
BB3
?
BB4
BB5
BB6
BB7
BB8'
BB8
BB1
BB2
BB3
BB4
BB5
BB6
BB7
BB8'
BB8

Variable

L, the maximum virtual trace length.
(The number of branch predictions stored in each virtual trace.)

Variable

L, the maximum virtual trace length.
(The number of branch predictions stored in each virtual trace.)

- Defined by the size of the memory for virtual traces.
- $L = 1 \Rightarrow$ Trivial traces: like assuming all branches are *unknown*.
- $L > 1 \Rightarrow$ Non-trivial traces: speculation is used to reduce the cost of the predicted execution path.

Experiment 1

What is the relationship between trace length and WCET?

Experiment 1

What is the relationship between trace length and WCET?

- Compare $L = 1$ against $L \in [4, 8, 12, 16]$ for various benchmark programs.

Experiment 1

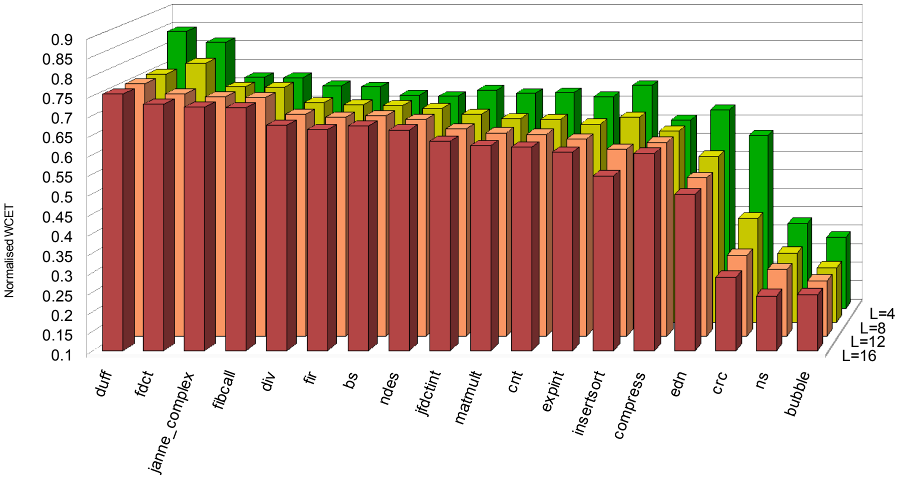
What is the relationship between trace length and WCET?

- Compare $L = 1$ against $L \in [4, 8, 12, 16]$ for various benchmark programs.
- Using experimental platform from:

J. Whitham. Real-time processor architectures for worst case execution time reduction. PhD Thesis YCST-2008-01, University of York, 2008.

Results 1

Normalized against results for $L = 1$.



Experiment 2

The Bodin-Puaut algorithm never changes any branch predictions once made... is an opportunity for improvement being lost?

Experiment 2

The Bodin-Puaut algorithm never changes any branch predictions once made... is an opportunity for improvement being lost?

1. Try flipping each branch prediction in each program,
   i.e. not taken $\leftrightarrow$ taken, and evaluate the new WCET in each case.

Experiment 2

The Bodin-Puaut algorithm never changes any branch predictions once made... is an opportunity for improvement being lost?

1. Try flipping each branch prediction in each program, i.e. not taken $\leftrightarrow$ taken, and evaluate the new WCET in each case.
2. If an improvement is found, repeat step 1.

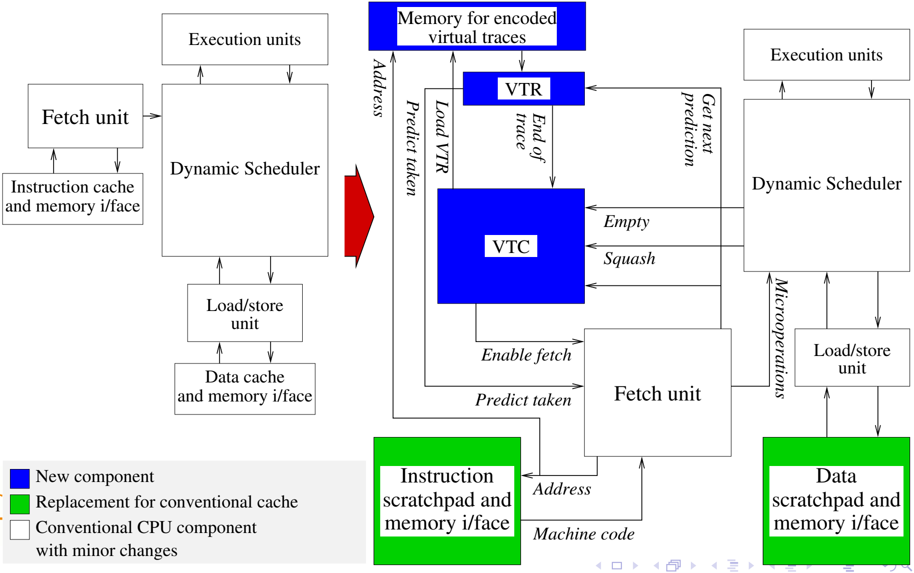
Results 2

Program	$L = 4$			$L = 16$		
	i	NW	%ch	i	NW	%ch
cnt	1	0.679	0.1%	1	0.618	0.1%
compress	3	0.604	0.2%	2	0.601	0.2%
edn	2	0.629	0.5%		n/a	
expint	1	0.668	0.0%	1	0.605	0.0%
fibcall	1	0.616	16.8%	1	0.616	16.8%
janne_complex	2	0.675	6.7%	2	0.675	6.7%
matmult	2	0.677	0.1%	2	0.622	0.1%
ndes	4	0.669	0.1%	5	0.661	0.2%
ns	3	0.305	7.9%	2	0.198	21.3%

Related Work

- Scratchpads
  Puaut, Suhendra, Wehmeyer
- Single-path paradigm
  Puschner
- Hybrid timing analysis
  Mohan, Mueller
- Dataflow-like computing models
  Lee et al.

Implementation



- New component
- Replacement for conventional cache
- Conventional CPU component with minor changes

Conclusion

- Virtual traces improve on scratchpad traces by allowing an entire program to be executed in trace form with a low storage cost for the trace data.

Conclusion

- Virtual traces improve on scratchpad traces by allowing an entire program to be executed in trace form with a low storage cost for the trace data.
- Consequently, greater WCET reductions are possible.

Conclusion

- Virtual traces improve on scratchpad traces by allowing an entire program to be executed in trace form with a low storage cost for the trace data.
- Consequently, greater WCET reductions are possible.
- The greatest benefits of traces are seen when one execution path is much more costly than the others.

Conclusion

- Virtual traces improve on scratchpad traces by allowing an entire program to be executed in trace form with a low storage cost for the trace data.
- Consequently, greater WCET reductions are possible.
- The greatest benefits of traces are seen when one execution path is much more costly than the others.
- Limiting virtual trace sizes to $L = 8$ is sufficient in many cases.

Conclusion

- Virtual traces improve on scratchpad traces by allowing an entire program to be executed in trace form with a low storage cost for the trace data.
- Consequently, greater WCET reductions are possible.
- The greatest benefits of traces are seen when one execution path is much more costly than the others.
- Limiting virtual trace sizes to $L = 8$ is sufficient in many cases.
- The Bodin-Puaut algorithm is not optimal but only minor improvements are possible.

End

- All questions and comments are welcome!
- Further information:
  `http://www.jwhitham.org.uk/pubs/`