Predictable Out-of-order Execution Using Virtual Traces

Jack Whitham and Neil Audsley

December 3rd 2008

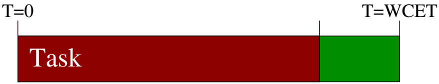
http://www.jwhitham.org.uk/c/vt.html

Topics in this talk

1. General issues with state-of-the-art worst case execution time (WCET) analysis.
2. Problem: design a CPU to reduce the WCET of a task.
3. Traces; a solution.
4. Virtual traces; a further improvement.
5. Experiments, results, observations.
6. Data scratchpads; a problem.
7. Conclusion.

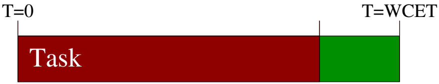
State-of-the-art WCET analysis

Requirement: find an upper bound on the execution time of a task: C, the WCET.



State-of-the-art WCET analysis

Requirement: find an upper bound on the execution time of a task: C, the WCET.

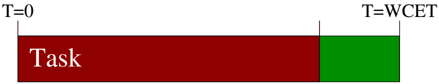


Solution:

- Model the functional properties of the code.
- Model interactions between code and CPU hardware features.

State-of-the-art WCET analysis

Requirement: find an upper bound on the execution time of a task: C, the WCET.

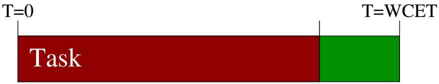


Solution:

- Model the functional properties of the code.
- Model interactions between code and CPU hardware features.
  → This is the focus of my talk; examples include:
  - cache modeling
    e.g. determine how often a load operation X "hits"

State-of-the-art WCET analysis

Requirement: find an upper bound on the execution time of a task: C, the WCET.



Solution:

- Model the functional properties of the code.
- Model interactions between code and CPU hardware features.
  → This is the focus of my talk; examples include:
  - cache modeling
    e.g. determine how often a load operation X "hits"
  - pipeline modeling
    e.g. determine the worst-case state of the pipeline at point Y

State-of-the-art WCET analysis

For complex CPUs, there are (*at least*) three problems:

State-of-the-art WCET analysis

For complex CPUs, there are (*at least*) three problems:

- Any CPU feature can be modeled *in isolation*, but since the features *interact*, this is *insufficient to capture worst-case behavior*.

State-of-the-art WCET analysis

For complex CPUs, there are (*at least*) three problems:

- Any CPU feature can be modeled *in isolation*, but since the features *interact*, this is *insufficient to capture worst-case behavior*.
- CPU manufacturers cannot usually provide exact specifications for their CPUs, but highly accurate data is *required* to make the models.

State-of-the-art WCET analysis

For complex CPUs, there are (*at least*) three problems:

- Any CPU feature can be modeled *in isolation*, but since the features *interact*, this is *insufficient to capture worst-case behavior*.
- CPU manufacturers cannot usually provide exact specifications for their CPUs, but highly accurate data is *required* to make the models.
- The *timing anomaly problem* makes it impossible to determine which hardware state leads to a greater execution time.

> **Timing anomaly:** A locally smaller WCET may lead to a globally greater WCET.

State-of-the-art WCET analysis

For complex CPUs, there are (*at least*) three problems:

- Any CPU feature can be modeled *in isolation*, but since the features *interact*, this is *insufficient to capture worst-case behavior*.
- CPU manufacturers cannot usually provide exact specifications for their CPUs, but highly accurate data is *required* to make the models.
- The *timing anomaly problem* makes it impossible to determine which hardware state leads to a greater execution time.

> **Timing anomaly:** A locally smaller WCET may lead to a globally greater WCET.

Solutions do exist for all of these problems, but they (1) raise the engineering cost and/or (2) increase the WCET.

A new sort of solution

Suppose we can replace the CPU with a new (or updated) design, aiming to (1) support WCET analysis, and (2) allow the WCET to be reduced.

A new sort of solution

Suppose we can replace the CPU with a new (or updated) design, aiming to (1) support WCET analysis, and (2) allow the WCET to be reduced.

How could WCET analysis be supported?

- Allow the CPU behavior to be safely captured by measurement.

A new sort of solution

Suppose we can replace the CPU with a new (or updated) design, aiming to (1) support WCET analysis, and (2) allow the WCET to be reduced.

How could WCET analysis be supported?

- Allow the CPU behavior to be safely captured by measurement. *Not usually possible in current CPUs, since there are too many factors affecting timing, and the CPU can't be forced into a known state.*

A new sort of solution

Suppose we can replace the CPU with a new (or updated) design, aiming to (1) support WCET analysis, and (2) allow the WCET to be reduced.

How could WCET analysis be supported?

- Allow the CPU behavior to be safely captured by measurement. *Not usually possible in current CPUs, since there are too many factors affecting timing, and the CPU can't be forced into a known state.*
- Constrain/isolate some parts of the design to simplify modeling by reducing possible interactions.

A new sort of solution

Suppose we can replace the CPU with a new (or updated) design, aiming to (1) support WCET analysis, and (2) allow the WCET to be reduced.

How could WCET analysis be supported?

- Allow the CPU behavior to be safely captured by measurement. *Not usually possible in current CPUs, since there are too many factors affecting timing, and the CPU can't be forced into a known state.*
- Constrain/isolate some parts of the design to simplify modeling by reducing possible interactions.

How could the WCET be reduced?

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

A new sort of solution

Suppose we can replace the CPU with a new (or updated) design, aiming to (1) support WCET analysis, and (2) allow the WCET to be reduced.

How could WCET analysis be supported?

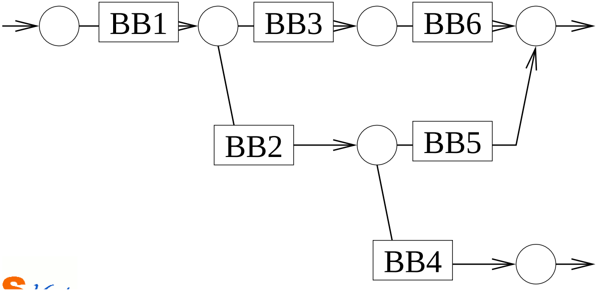
- Allow the CPU behavior to be safely captured by measurement. *Not usually possible in current CPUs, since there are too many factors affecting timing, and the CPU can't be forced into a known state.*
- Constrain/isolate some parts of the design to simplify modeling by reducing possible interactions.

How could the WCET be reduced?

- Accommodate speculative and superscalar out-of-order operation so that throughput can be increased versus a simple CPU.
- Reduce pessimism in the WCET model.

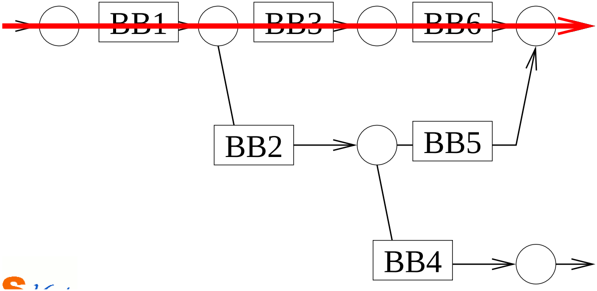
One possibility: a trace

A trace is a path through a program, chosen to reduce C, the worst-case execution time.



One possibility: a trace

A trace is a path through a program, chosen to reduce C, the worst-case execution time.



How does this help?

If a program is composed of *traces*, analysis can ignore *how* a computation is performed by the CPU, and instead consider only one thing:

> The length of time taken for each path through each trace.

How does this help?

If a program is composed of *traces*, analysis can ignore *how* a computation is performed by the CPU, and instead consider only one thing:

> The length of time taken for each path through each trace.

If a trace contains n branches, then there are $n + 1$ paths through it.

How does this help?

If a program is composed of *traces*, analysis can ignore *how* a computation is performed by the CPU, and instead consider only one thing:

> The length of time taken for each path through each trace.

If a trace contains n branches, then there are $n + 1$ paths through it.
$\Rightarrow$ There are exactly $n + 1$ ways that it could ever be executed.

How does this help?

If a program is composed of *traces*, analysis can ignore *how* a computation is performed by the CPU, and instead consider only one thing:

> The length of time taken for each path through each trace.

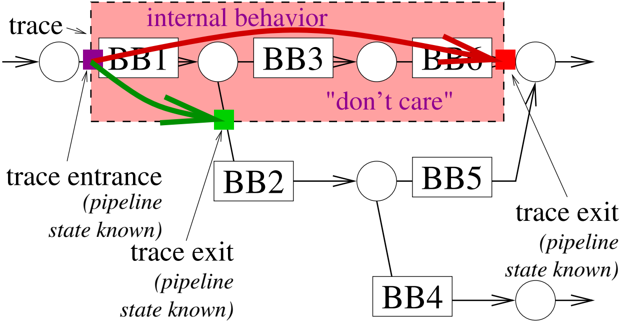
If a trace contains n branches, then there are $n + 1$ paths through it.
$\Rightarrow$ There are exactly $n + 1$ ways that it could ever be executed.
$\Rightarrow$ There are exactly $n + 1$ sequences of pipeline states.

How does this help?

The intermediate pipeline states are irrelevant to analysis.

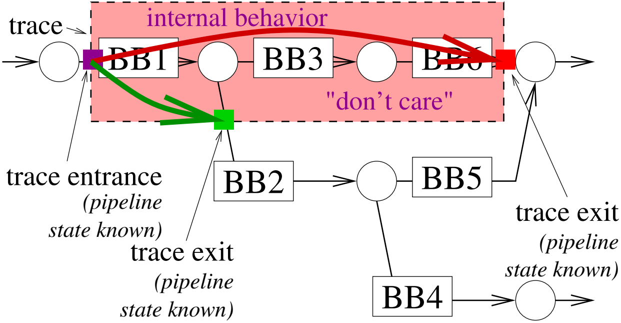
How does this help?

The intermediate pipeline states are irrelevant to analysis.



How does this help?

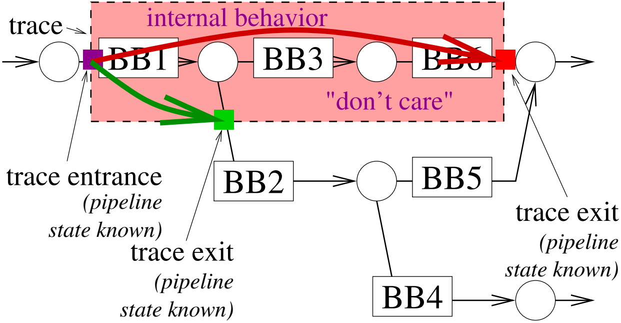
The intermediate pipeline states are irrelevant to analysis.



- The trace begins and ends in a known pipeline state.

How does this help?

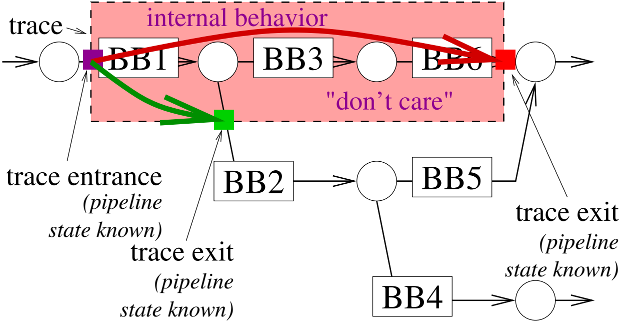
The intermediate pipeline states are irrelevant to analysis.



- The trace begins and ends in a known pipeline state.
- The total time for each path is exactly known (it can be measured).

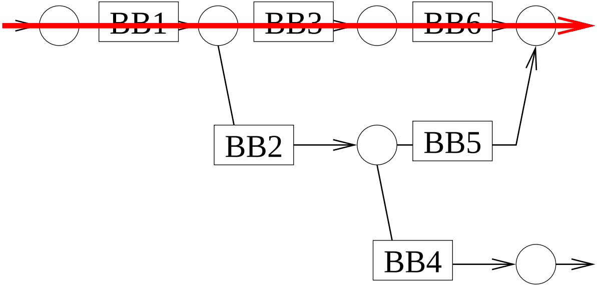
How does this help?

The intermediate pipeline states are irrelevant to analysis.



- The trace begins and ends in a known pipeline state.
- The total time for each path is exactly known (it can be measured).
- **The result:** speculation and superscalar out-of-order execution don't have to be modeled!

Static branch prediction



Contrast with *static branch prediction*. With a virtual trace, the main path has a well-defined end point, so the number of possible pipeline states is *bounded*. Static branch prediction omits this important restriction.

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

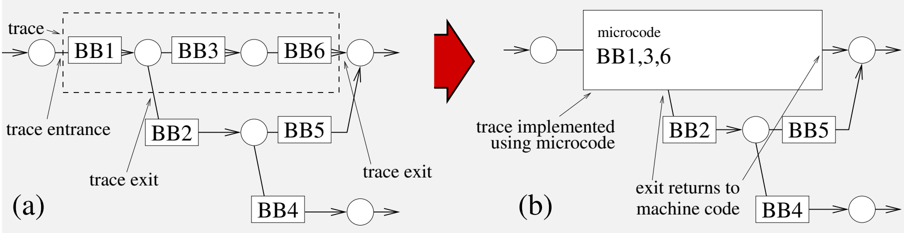
In previous work, we considered the use of a *trace scratchpad* to implement traces and meet the requirements, used as follows:

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 *optimize execution for one path*.

Previous work

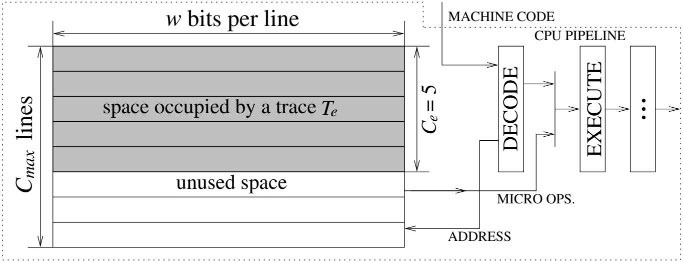
In previous work, we considered the use of a *trace scratchpad* to implement traces and meet the requirements, used as follows:

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 *optimize 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 equivalent to traces within the WCET analysis model, but some practical problems are solved:

- the need for a custom CPU with a writable microcode store,

Virtual trace

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

Virtual traces are equivalent to traces within the WCET analysis model, 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,

Virtual trace

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

Virtual traces are equivalent to traces within the WCET analysis model, 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 memory space requirements of microcode.

Virtual trace

The virtual trace controls a conventional *but constrained* dynamic CPU scheduler.

Virtual trace

The virtual trace controls a conventional *but constrained* dynamic CPU scheduler.

We regard the CPU dynamic scheduler as a decoder:

machine code + virtual trace → “microcode”

Virtual trace

The virtual trace controls a conventional *but constrained* dynamic CPU scheduler.

We regard the CPU dynamic scheduler as a decoder:

machine code + virtual trace → “microcode”

Virtual in the sense that the microcode is generated dynamically - we know what the scheduler will do, but we don’t explicitly encode it.

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.
   Solution: Constrain execution to a known state at the beginning and end of each trace.
   The effects of all interactions are captured when the virtual trace execution time is measured.

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.
   Solution: Constrain execution to a known state at the beginning and end of each trace.
   The effects of all interactions are captured when the virtual trace execution time is measured.
2. **Problem:** CPUs are hard to model.

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.
   Solution: Constrain execution to a known state at the beginning and end of each trace.
   The effects of all interactions are captured when the virtual trace execution time is measured.
2. **Problem:** CPUs are hard to model.
   Solution: Don't try to model the CPU.
   Instead, *constrain* CPU behavior so that it can be *measured* safely.

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.
   Solution: Constrain execution to a known state at the beginning and end of each trace.
   The effects of all interactions are captured when the virtual trace execution time is measured.
2. **Problem:** CPUs are hard to model.
   Solution: Don't try to model the CPU.
   Instead, *constrain* CPU behavior so that it can be *measured* safely.
3. **Problem:** Timing anomalies can occur in complex CPUs.

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.
   Solution: Constrain execution to a known state at the beginning and end of each trace.
   The effects of all interactions are captured when the virtual trace execution time is measured.
2. **Problem:** CPUs are hard to model.
   Solution: Don't try to model the CPU.
   Instead, *constrain* CPU behavior so that it can be *measured* safely.
3. **Problem:** Timing anomalies can occur in complex CPUs.
   Solution: Because of resynchronization, timing anomalies can't propagate from one trace to another.

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.
   Solution: Constrain execution to a known state at the beginning and end of each trace.
   The effects of all interactions are captured when the virtual trace execution time is measured.
2. **Problem:** CPUs are hard to model.
   Solution: Don't try to model the CPU.
   Instead, *constrain* CPU behavior so that it can be *measured* safely.
3. **Problem:** Timing anomalies can occur in complex CPUs.
   Solution: Because of resynchronization, timing anomalies can't propagate from one trace to another.
4. **Problem:** How can the WCET be reduced?

This helps!

How are the issues of WCET analysis addressed?

1. **Problem:** Pipeline components can interact.
   Solution: Constrain execution to a known state at the beginning and end of each trace.
   The effects of all interactions are captured when the virtual trace execution time is measured.
2. **Problem:** CPUs are hard to model.
   Solution: Don't try to model the CPU.
   Instead, *constrain* CPU behavior so that it can be *measured* safely.
3. **Problem:** Timing anomalies can occur in complex CPUs.
   Solution: Because of resynchronization, timing anomalies can't propagate from one trace to another.
4. **Problem:** How can the WCET be reduced?
   Solution: Allow speculative and out-of-order execution within a trace.

Statement, research questions

If a program runs using virtual traces,
and program functionality can be modeled,
then an exact bound for the WCET C can be found.

Statement, research questions

If a program runs using virtual traces,
and program functionality can be modeled,
then an exact bound for the WCET C can be found.

- Q1: Given a task T, is C lower if
  (a) a simple in-order CPU is used (minimum one CPI), or
  (b) a virtual trace CPU is used.

Statement, research questions

If a program runs using virtual traces,
and program functionality can be modeled,
then an exact bound for the WCET C can be found.

- Q1: Given a task T, is C lower if
  (a) a simple in-order CPU is used (minimum one CPI), or
  (b) a virtual trace CPU is used.
- Q2: Which of the constraints needed to implement virtual traces have the greatest effect on execution time?

In this work

- Virtual traces are implemented using the M5 O3 CPU simulator.

In this work

- Virtual traces are implemented using the M5 O3 CPU simulator.
  M5 - Architecture simulator from UMich.
  O3 - **O**ut **O**f **O**rder CPU.

In this work

- Virtual traces are implemented using the M5 O3 CPU simulator.
  M5 - Architecture simulator from UMich.
  O3 - **O**ut **O**f **O**rder CPU.
- Memory subsystem is assumed to be deterministic: *perfect caches*.

In this work

- Virtual traces are implemented using the M5 O3 CPU simulator.
  M5 - Architecture simulator from UMich.
  O3 - **O**ut **O**f **O**rder CPU.
- Memory subsystem is assumed to be deterministic: *perfect caches*.
- The two research questions (comparison with in-order CPU, effects of constraints) are investigated.

In this work

- Virtual traces are implemented using the M5 O3 CPU simulator.
  M5 - Architecture simulator from UMich.
  O3 - **O**ut **O**f **O**rder CPU.
- Memory subsystem is assumed to be deterministic: *perfect caches*.
- The two research questions (comparison with in-order CPU, effects of constraints) are investigated.
- Virtual traces are assigned to a single-path program using profiling.
  - In reality, WCET analysis would be used.

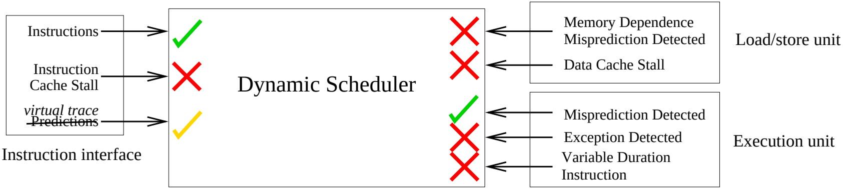
In this work

- Virtual traces are implemented using the M5 O3 CPU simulator.
  M5 - Architecture simulator from UMich.
  O3 - **O**ut **O**f **O**rder CPU.
- Memory subsystem is assumed to be deterministic: *perfect caches*.
- The two research questions (comparison with in-order CPU, effects of constraints) are investigated.
- Virtual traces are assigned to a single-path program using profiling.
  - In reality, WCET analysis would be used.
  - Chicken and egg problem!

J. Whitham and N. Audsley, Forming Virtual Traces for WCET Analysis and Reduction, Proc. RTCSA, 377–386, 2008.

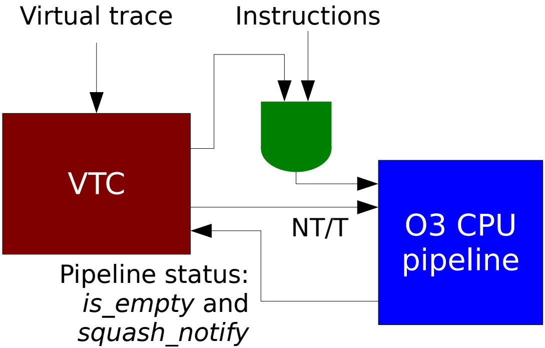
How virtual traces are implemented

(1) Sources of *timing noise* in O3 are constrained or eliminated:



How virtual traces are implemented

(2) The *virtual trace controller* (VTC) generates branch predictions and manages the flow of instructions into the pipeline:



Result: O3+VTC CPU: O3 with virtual trace extensions.

Experiment 1

Q1: Given a task T, is C lower if
(a) a simple in-order CPU is used (minimum one CPI), or
(b) a virtual trace CPU is used.

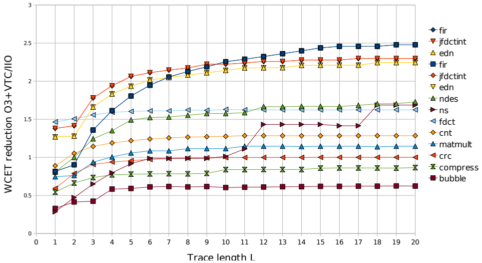
A subset of the Mälardalen benchmarks were executed within the following environments, measuring execution time:

- **IIO**: Idealized in-order CPU.
  Exactly one instruction executed every clock cycle.
- **O3+VTC**: Virtual trace CPU with maximum trace length $L \in [1, 20]$.

Results 1

	IIO	O3+VTC
bs	92	79
bubble	5,286	8,454
cnt	3,580	2,786
compress	3,545	4,093
crc	21,096	21,082
duff	496	509
edn	97,001	43,227
expint	533	385
fdct	3,410	2,093
fibcall	44	26
fir	2,988	1,206
insertsort	887	676
janne	348	299
jfdctint	3,467	1,509
matmult	142,810	124,595
ndes	40,284	23,294
ns	2,852	1,691

O3+VTC: Some WCET reduction achieved for 14 of 17 cases, up to 2.5×.

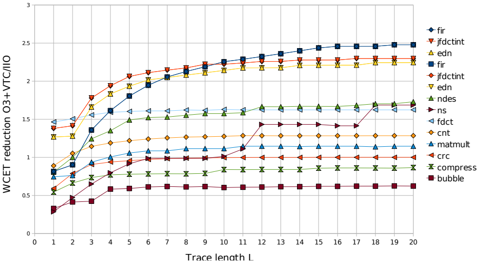


Results 1

	IIO	O3+VTC
bs	92	79
bubble	5,286	8,454
cnt	3,580	2,786
compress	3,545	4,093
crc	21,096	21,082
duff	496	509
edn	97,001	43,227
expint	533	385
fdct	3,410	2,093
fibcall	44	26
fir	2,988	1,206
insertsort	887	676
janne	348	299
jfdctint	3,467	1,509
matmult	142,810	124,595
ndes	40,284	23,294
ns	2,852	1,691

O3+VTC: Some WCET reduction achieved for 14 of 17 cases, up to 2.5×.

But available WCET reductions are highly dependent on program structure; unpredictable branches are a problem. *If-conversion* is a solution (localized single-path programming).



Experiment 2

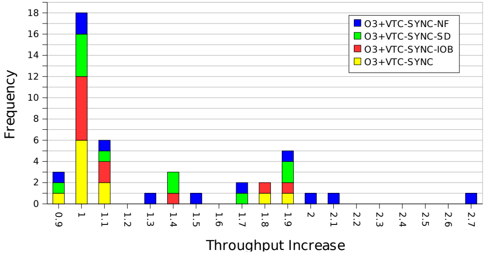
> Q2: Which of the constraints needed to implement virtual traces have the greatest effect on execution time?

A subset of the Mälardalen benchmarks were executed within the following environments, measuring execution time. Each environment removes one of the constraints of O3+VTC:

- **O3+VTC-SYNC**: The pipeline is not resynchronized at trace end.
- **O3+VTC-SYNC-IOB**: Branches may be executed out of order.
- **O3+VTC-SYNC-SD**: Dynamic memory disambiguation is used.
- **O3+VTC-SYNC-NF**: Dynamic memory forwarding is permitted.

Results 2

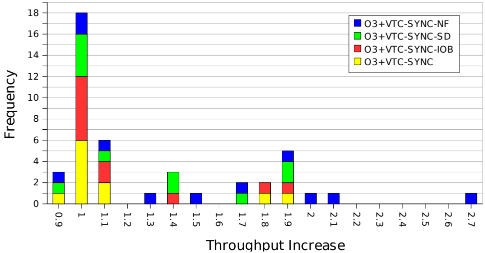
Unpredictable branches are only one problem! CPU constraints also increase C: O3+VTC is up to 3.6× slower than O3.



Results 2

Unpredictable branches are only one problem! CPU constraints also increase C: O3+VTC is up to 3.6× slower than O3.

In terms of C, the most severe constraints are those preventing dynamic memory *disambiguation* and *forwarding*.

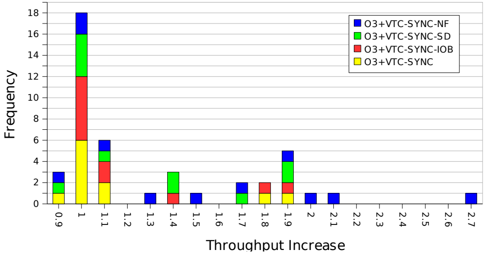


Results 2

Unpredictable branches are only one problem! CPU constraints also increase C: O3+VTC is up to 3.6× slower than O3.

In terms of C, the most severe constraints are those preventing dynamic memory *disambiguation* and *forwarding*.

If only we could predict the addresses of loads and stores!



Further Investigation

In fact, the memory addresses used for data accesses have an even more significant effect on performance, because of the time taken to fetch data.

Further Investigation

In fact, the memory addresses used for data accesses have an even more significant effect on performance, because of the time taken to fetch data.

- This work assumed a *perfect data cache*. A real data cache would be a source of timing noise; this would be a severe problem. Every hit/miss needs to be predicted in advance.

Further Investigation

In fact, the memory addresses used for data accesses have an even more significant effect on performance, because of the time taken to fetch data.

- This work assumed a *perfect data cache*. A real data cache would be a source of timing noise; this would be a severe problem. Every hit/miss needs to be predicted in advance.
- A scratchpad memory or locked cache would need to be used. Automatic data scratchpad allocation is *necessary* but *difficult*.

Further Investigation

In fact, the memory addresses used for data accesses have an even more significant effect on performance, because of the time taken to fetch data.

- This work assumed a *perfect data cache*. A real data cache would be a source of timing noise; this would be a severe problem. Every hit/miss needs to be predicted in advance.
- A scratchpad memory or locked cache would need to be used. Automatic data scratchpad allocation is *necessary* but *difficult*.
    - Q1: Which variables should be stored in scratchpad? Which should be stored in main memory?

Further Investigation

In fact, the memory addresses used for data accesses have an even more significant effect on performance, because of the time taken to fetch data.

- This work assumed a *perfect data cache*. A real data cache would be a source of timing noise; this would be a severe problem. Every hit/miss needs to be predicted in advance.
- A scratchpad memory or locked cache would need to be used. Automatic data scratchpad allocation is *necessary* but *difficult*.
  - Q1: Which variables should be stored in scratchpad? Which should be stored in main memory?
  - Q2: How often should the partition be changed?

Further Investigation

In fact, the memory addresses used for data accesses have an even more significant effect on performance, because of the time taken to fetch data.

- This work assumed a *perfect data cache*. A real data cache would be a source of timing noise; this would be a severe problem. Every hit/miss needs to be predicted in advance.
- A scratchpad memory or locked cache would need to be used. Automatic data scratchpad allocation is *necessary* but *difficult*.
  - Q1: Which variables should be stored in scratchpad? Which should be stored in main memory?
  - Q2: How often should the partition be changed?

A hard problem for general C code:

- Pointers can have almost any value.
- Memory might be allocated dynamically.

The same problems affect data cache modeling.

Further Investigation

A solution for automatic data scratchpad allocation would also relax the constraints on dynamic data accesses, because:

Further Investigation

A solution for automatic data scratchpad allocation would also relax the constraints on dynamic data accesses, because:

- The range of addresses for each data access would be known, so dynamic disambiguation would be unnecessary.

Further Investigation

A solution for automatic data scratchpad allocation would also relax the constraints on dynamic data accesses, because:

- The range of addresses for each data access would be known, so dynamic disambiguation would be unnecessary.

However, this is not an easy problem.

- Existing automatic data scratchpad allocation systems assume no pointers (global/stack arrays only).

Further Investigation

A solution for automatic data scratchpad allocation would also relax the constraints on dynamic data accesses, because:

- The range of addresses for each data access would be known, so dynamic disambiguation would be unnecessary.

However, this is not an easy problem.

- Existing automatic data scratchpad allocation systems assume no pointers (global/stack arrays only).
- Other solutions assume an unconventional program paradigm (e.g. dataflow/actor-oriented model).

Further Investigation

A solution for automatic data scratchpad allocation would also relax the constraints on dynamic data accesses, because:

- The range of addresses for each data access would be known, so dynamic disambiguation would be unnecessary.

However, this is not an easy problem.

- Existing automatic data scratchpad allocation systems assume no pointers (global/stack arrays only).
- Other solutions assume an unconventional program paradigm (e.g. dataflow/actor-oriented model).

The problem needs to be solved for typical C programs; otherwise, assumptions such as “perfect data cache” (as made in this work) will continue to be unrealistic.

Conclusions

- Virtual traces allow speculative and out-of-order execution to be used, so C can be reduced in comparison to an in-order CPU design.

Conclusions

- Virtual traces allow speculative and out-of-order execution to be used, so C can be reduced in comparison to an in-order CPU design.
- The CPU constraints reduce the maximum performance but increase the guaranteed performance.

Conclusions

- Virtual traces allow speculative and out-of-order execution to be used, so C can be reduced in comparison to an in-order CPU design.
- The CPU constraints reduce the maximum performance but increase the guaranteed performance.
- Predictable management of data accesses is a problem that saps the performance of virtual traces.

Conclusions

- Virtual traces allow speculative and out-of-order execution to be used, so C can be reduced in comparison to an in-order CPU design.
- The CPU constraints reduce the maximum performance but increase the guaranteed performance.
- Predictable management of data accesses is a problem that saps the performance of virtual traces.
- The automatic data scratchpad allocation problem must be solved.

End

- All questions and comments are welcome!
- You can find the O3+VTC experimental software on the web at `http://www.jwhitham.org.uk/c/vt.html`