What now?

SVP, Microgrids, MGSim,
"hardware support for concurrency management"

R. Poss, Oct. 2012
Universiteit van Amsterdam

Original research question

Original research question

- Can hardware multithreading be made cheaper and at least as effective than OoOE at tolerating fine-grained latencies on single cores, including memory loads and FPUs?

Original research question

- Can hardware multithreading be made cheaper and at least as effective than OoOE at tolerating fine-grained latencies on single cores, including memory loads and FPUs?
  - Answer: yes!

Original research question

- Can hardware multithreading be made cheaper and at least as effective than OoOE at tolerating fine-grained latencies on single cores, including memory loads and FPUs?
  - Answer: yes!
  - Long answer: yes, but...

Original research question

- Can hardware multithreading be made cheaper and at least as effective than OoOE at tolerating fine-grained latencies on single cores, including memory loads and FPUs?
  - Answer: yes!
  - Long answer: yes, but...
  - It's been 10 years, and we're still working on that one. But so are others!

1996-2009, AT A GLANCE

latency tolerance

1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

D-RISC

1996

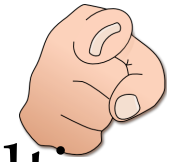
1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

D-RISC

1996



multi-cores!

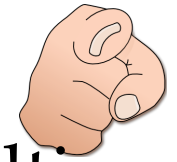
1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

D-RISC

1996



multi-cores!

hardware
concurrency
management

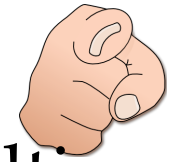
1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

D-RISC

1996



multi-cores!

hardware
concurrency
management

Microgrids

2004

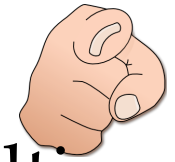
1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

D-RISC

1996



multi-cores!

hardware
concurrency
management

Microgrids

2004

programming
models!

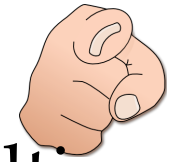
1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

D-RISC

1996



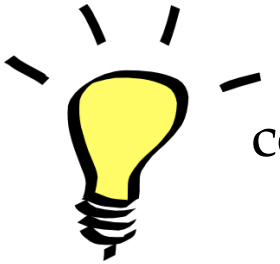
multi-cores!

hardware
concurrency
management

Microgrids

2004

programming
models!



abstract
concurrency
resources

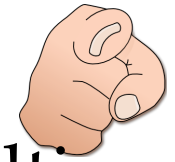
1996-2009, AT A GLANCE

latency tolerance

dataflow
scheduling

D-RISC

1996



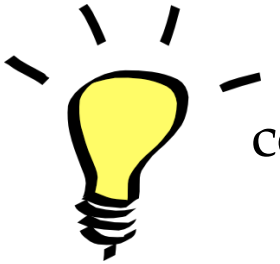
multi-cores!

hardware
concurrency
management

Microgrids

2004

programming
models!



abstract
concurrency
resources

SVP

2009

Updated research question (2008)

Updated research question (2008)

- Can hardware concurrency management provide more efficient and cost-attractive support than software-based approaches for latency tolerance in general-purpose concurrent code running on multi-cores?

Updated research question (2008)

- Can hardware concurrency management provide more efficient and cost-attractive support than software-based approaches for latency tolerance in general-purpose concurrent code running on multi-cores?
  - Answer: yes!

Updated research question (2008)

- Can hardware concurrency management provide more efficient and cost-attractive support than software-based approaches for latency tolerance in general-purpose concurrent code running on multi-cores?
  - Answer: yes!
  - Long answer: yes if combined with DRISC.

Updated research question (2008)

- Can hardware concurrency management provide more efficient and cost-attractive support than software-based approaches for latency tolerance in general-purpose concurrent code running on multi-cores?
  - Answer: yes!
  - Long answer: yes if combined with DRISC.
  - "Generality" is still poorly understood. How general is DRISC?

2008-2012, at a glance

SVP

Microgrids

D-RISC

2008-2012, at a glance

SVP

Microgrids

D-RISC

memory scalability!

2008-2012, AT A GLANCE

SVP

Microgrids

D-RISC

2008

memory scalability!

migration to point of last use

2008-2012, AT A GLANCE

SVP

Microgrids

D-RISC

2008

memory scalability!

migration to point of last use

Diffusion protocol

Jony, Mike, Qiang

2008-2012, AT A GLANCE

SVP

Microgrids

D-RISC

2008

memory scalability!

migration to point of last use

use concurrency events as coherency events

Diffusion protocol

Jony, Mike, Qiang

2008-2012, AT A GLANCE

SVP

Microgrids

D-RISC

2008

memory scalability!

migration to point of last use

Diffusion protocol

Jony, Mike, Qiang

use concurrency events as coherency events

Bulk consistency

Jony, Andrei, kena

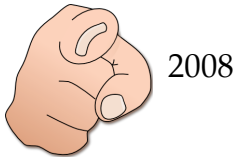
2008-2012, AT A GLANCE

SVP

Microgrids

D-RISC

work placement!



memory scalability!

migration to point of last use

Diffusion protocol

Jony, Mike, Qiang

use concurrency events as coherency events

Bulk consistency

Jony, Andrei, kena

2008-2012, AT A GLANCE

SVP

Microgrids

D-RISC

work placement!

memory scalability!

designate clusters of cores as single addresses

migration to point of last use

use concurrency events as coherency events

Diffusion protocol

Jony, Mike, Qiang

Bulk consistency

Jony, Andrei, kena

2008-2012, AT A GLANCE

SVP

Microgrids

D-RISC

work placement!

memory scalability!

designate clusters of cores as single addresses

Resource models

Michiel, Mike, kena, Clemens

migration to point of last use

Diffusion protocol

Jony, Mike, Qiang

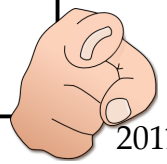
use concurrency events as coherency events

Bulk consistency

Jony, Andrei, kena

2008-2012, AT A GLANCE

SVP



2011

Microgrids

D-RISC

cost models!

2010

work placement!

2008

memory scalability!

designate clusters of cores as single addresses

Resource models

Michiel, Mike, kena, Clemens

migration to point of last use

Diffusion protocol

Jony, Mike, Qiang

use concurrency events as coherency events

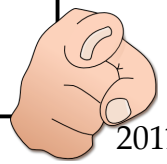
Bulk consistency

Jony, Andrei, kena

2008-2012, AT A GLANCE

2011

SVP

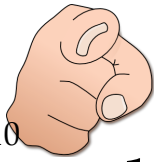


2011

Microgrids

D-RISC

generality!



2010

work placement!

cost models!

2008

memory scalability!

designate clusters of cores as single addresses

Resource models

Michiel, Mike, kena, Clemens

migration to point of last use

Diffusion protocol

Jony, Mike, Qiang

use concurrency events as coherency events

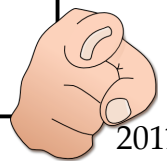
Bulk consistency

Jony, Andrei, kena

2008-2012, AT A GLANCE


2011

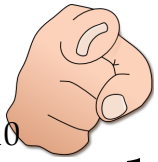
SVP


2011

Microgrids

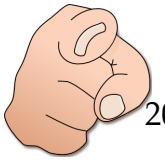
D-RISC

generality!


2010

cost models!


2008


2011

virtualizability!

work placement!

memory scalability!

designate clusters of cores as single addresses

migration to point of last use

use concurrency events as coherency events

Resource models

Michiel, Mike, kena, Clemens

Diffusion protocol

Jony, Mike, Qiang

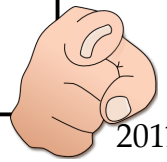
Bulk consistency

Jony, Andrei, kena

2008-2012, AT A GLANCE


2011

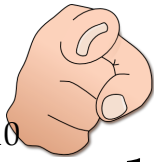
SVP


2011

Microgrids

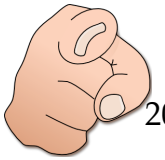
D-RISC

generality!


2010

cost models!

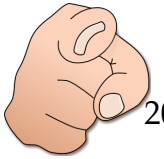

2008


2011

virtualizability!

work placement!

memory scalability!


2011

fault tolerance!


designate clusters of cores as single addresses


migration to point of last use


use concurrency events as coherency events

Resource models

Michiel, Mike, kena, Clemens

Diffusion protocol

Jony, Mike, Qiang

Bulk consistency

Jony, Andrei, kena

Ongoing research topics

Microgrids

Ongoing research topics

Microgrids

memory scalability!

Qiang

Beijing, ESA

Ongoing research topics

Microgrids

memory scalability!

Qiang

Beijing, ESA

fault tolerance!

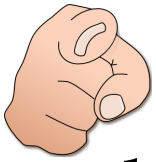
Jian

Beijing, ESA

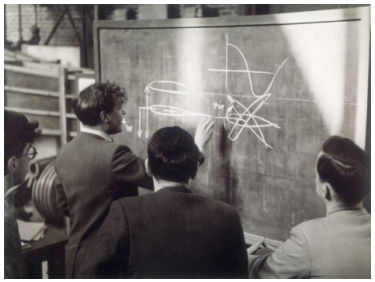
Ongoing research topics

SVP

Ongoing research topics



work placement!



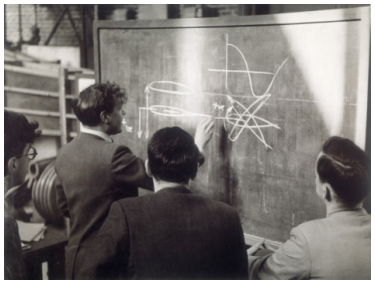
ADVANCE consortium

EU (ADVANCE)

SVP

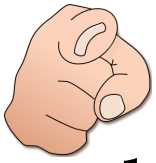
Ongoing research topics

cost models!

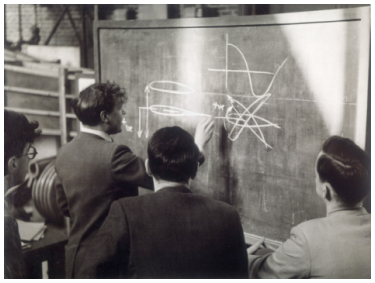


ADVANCE consortium

EU (ADVANCE)



work placement!



ADVANCE consortium

EU (ADVANCE)

SVP

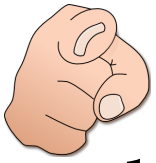
Ongoing research topics

Microgrids

SVP

Ongoing research topics

Microgrids



generality!

kena

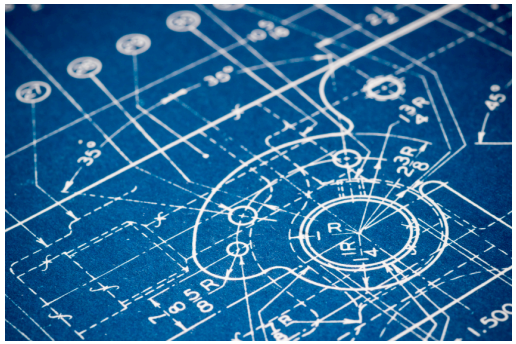
UvA

virtualizability!

SVP

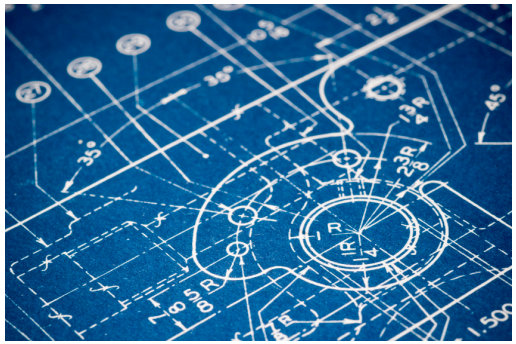
When the dust settles...

(on the theory side)



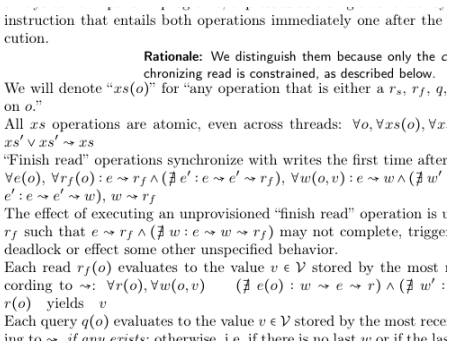
D-RISC “blue prints”

(UTLEON3, Mike’s theses,
MGSim’s core model)



D-RISC “blue prints”

(UTLEON3, Mike’s theses,
MGSim’s core model)



instruction that entails both operations immediately one after the
cution.

Rationale: We distinguish them because only the c
chronizing read is constrained, as described below.

4 We will denote “$xs(o)$” for “any operation that is either a r_s, r_f, q,
on o.”

5 All xs operations are atomic, even across threads: $\forall o, \forall xs(o), \forall x$
$xs' \vee xs' \rightsquigarrow xs$

6 “Finish read” operations synchronize with writes the first time after
$\forall e(o)$, $\forall r_f(o) : e \rightsquigarrow r_f \wedge (\nexists e' : e \rightsquigarrow e' \rightsquigarrow r_f)$, $\forall w(o,v) : e \rightsquigarrow w \wedge (\nexists w'$
$e' : e \rightsquigarrow e' \rightsquigarrow w)$, $w \rightsquigarrow r_f$

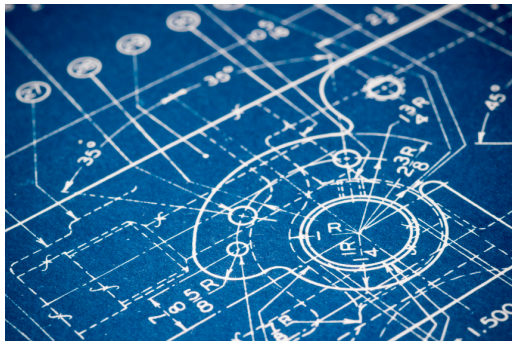
7 The effect of executing an unprovisioned “finish read” operation is u
r_f such that $e \rightsquigarrow r_f \wedge (\nexists w : e \rightsquigarrow w \rightsquigarrow r_f)$ may not complete, trigge
deadlock or effect some other unspecified behavior.

8 Each read $r_f(o)$ evaluates to the value $v \in \mathcal{V}$ stored by the most
cording to $\rightsquigarrow$: $\forall r(o), \forall w(o,v) \quad (\nexists e(o) : w \rightsquigarrow e \rightsquigarrow r) \wedge (\nexists w' :$
$r(o)$ yields v

9 Each query $q(o)$ evaluates to the value $v \in \mathcal{V}$ stored by the most rece
ing to $\rightsquigarrow$ *if any exists*; otherwise, i.e. if there is no last w or if the las

Protocol specifications

For concurrency management
and memory consistency in hardware



D-RISC “blue prints”

(UTLEON3, Mike’s theses,
MGSim’s core model)

instruction that entails both operations immediately one after the
cution.

Rationale: We distinguish them because only the c
chronizing read is constrained, as described below.

4 We will denote “$xs(o)$” for “any operation that is either a r_s, r_f, q,
on o.”

5 All xs operations are atomic, even across threads: $\forall o, \forall xs(o), \forall x$
$xs' \vee xs' \rightsquigarrow xs$

6 “Finish read” operations synchronize with writes the first time after
$\forall e(o)$, $\forall r_f(o) : e \rightsquigarrow r_f \wedge (\nexists e' : e \rightsquigarrow e' \rightsquigarrow r_f)$, $\forall w(o,v) : e \rightsquigarrow w \wedge (\nexists w'$
$e' : e \rightsquigarrow e' \rightsquigarrow w)$, $w \rightsquigarrow r_f$

7 The effect of executing an unprovisioned “finish read” operation is u
r_f such that $e \rightsquigarrow r_f \wedge (\nexists w : e \rightsquigarrow w \rightsquigarrow r_f)$ may not complete, trigge
deadlock or effect some other unspecified behavior.

8 Each read $r_f(o)$ evaluates to the value $v \in \mathcal{V}$ stored by the most
cording to $\rightsquigarrow$: $\forall r(o), \forall w(o,v) \quad (\nexists e(o) : w \rightsquigarrow e \rightsquigarrow r) \wedge (\nexists w' :$
$r(o)$ yields v

9 Each query $q(o)$ evaluates to the value $v \in \mathcal{V}$ stored by the most rece
ing to $\rightsquigarrow$ if any exists; otherwise, i.e. if there is no last w or if the las

Protocol specifications

For concurrency management
and memory consistency in hardware

I.5.9.1 Thread function definitions

Syntax

thread-function-definition:
`sl_def` (*identifier* [, *attributes*$_{opt}$ ¬
[, *thread-parameter-list*]$_{opt}$]$_{opt}$) ¬
compound-statement ¬
`sl_enddef`
external-declaration:
...
thread-function-definition

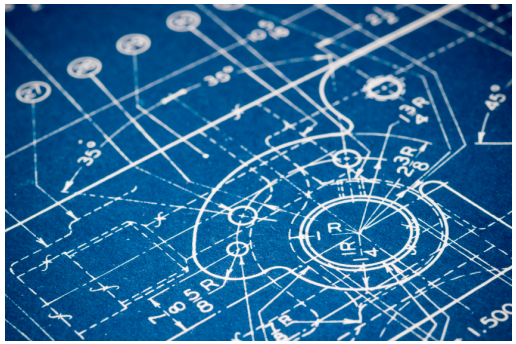
Constraints

1 The identifier in thread functions definitions is in the same name sp
and function names. Therefore, [II99, 6.9§3] and [II11b, 6.9§3] apply.

2 The constraints in Appendix I.5.7.6 apply.

Language specifications

For intermediate representations
able to target “any SVP implementation”



D-RISC “blue prints”

(UTLEON3, Mike’s theses,
MGSim’s core model)

instruction that entails both operations immediately one after the
cution.

Rationale: We distinguish them because only the c
chronizing read is constrained, as described below.

4 We will denote “$xs(o)$” for “any operation that is either a r_s, r_f, q,
on o.”

5 All xs operations are atomic, even across threads: $\forall o, \forall xs(o), \forall x$
$xs' \vee xs' \rightsquigarrow xs$

6 “Finish read” operations synchronize with writes the first time after
$\forall e(o)$, $\forall r_f(o) : e \rightsquigarrow r_f \wedge (\nexists e' : e \rightsquigarrow e' \rightsquigarrow r_f)$, $\forall w(o,v) : e \rightsquigarrow w \wedge (\nexists w'$
$e' : e \rightsquigarrow e' \rightsquigarrow w)$, $w \rightsquigarrow r_f$

7 The effect of executing an unprovisioned “finish read” operation is u
r_f such that $e \rightsquigarrow r_f \wedge (\nexists w : e \rightsquigarrow w \rightsquigarrow r_f)$ may not complete, trigge
deadlock or effect some other unspecified behavior.

8 Each read $r_f(o)$ evaluates to the value $v \in \mathcal{V}$ stored by the most
cording to $\rightsquigarrow$: $\forall r(o), \forall w(o,v) \quad (\nexists e(o) : w \rightsquigarrow e \rightsquigarrow r) \wedge (\nexists w' :$
$r(o)$ yields v

9 Each query $q(o)$ evaluates to the value $v \in \mathcal{V}$ stored by the most rece
ing to $\rightsquigarrow$ *if any exists*; otherwise, i.e. if there is no last w or if the las

Protocol specifications

For concurrency management
and memory consistency in hardware

I.5.9.1 Thread function definitions

Syntax

thread-function-definition:
sl_def (*identifier* [, *attributes*$_{opt}$ ¬
[, *thread-parameter-list*]$_{opt}$]$_{opt}$) ¬
compound-statement ¬
sl_enddef
external-declaration:
...
thread-function-definition

Constraints

1 The identifier in thread functions definitions is in the same name sp
and function names. Therefore, [II99, 6.9§3] and [II11b, 6.9§3] apply.

2 The constraints in Appendix I.5.7.6 apply.

Language specifications

For intermediate representations
able to target “any SVP implementation”

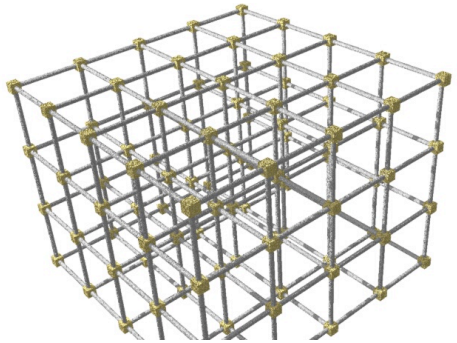
```
box NR : input.(fi), execenv.(e)
  ↦ output.(fo), expectedbehavior.(b) =
provided
 let $shape := { (Size, *, *) ∈ $fi }
               ∪ { (Dim, *) ∈ $fi },
 (CALReduce, ×, (Size), 1, $s) ∈ $shape
use
 $shape ⊂ $fo
 (Sequential) ∈ $e  ⟹ (TimeComplexity, $s) ∈ $b
 provided
  (DataParallelism, $p) ∈ $e
 use
  $s ≤ 50 ⟹ (TimeComplexity, $s) ∈ $b
  $s > 50 ⟹ (TimeComplexity, $s / $p) ∈ $b
 end
end;
```

Listing 7: Passport for box NR, with parallel behavior.

Resource models and operational semantics

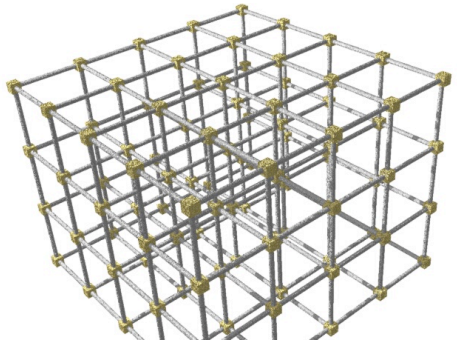
(Work in progress w/ UH)

(on the technology side)



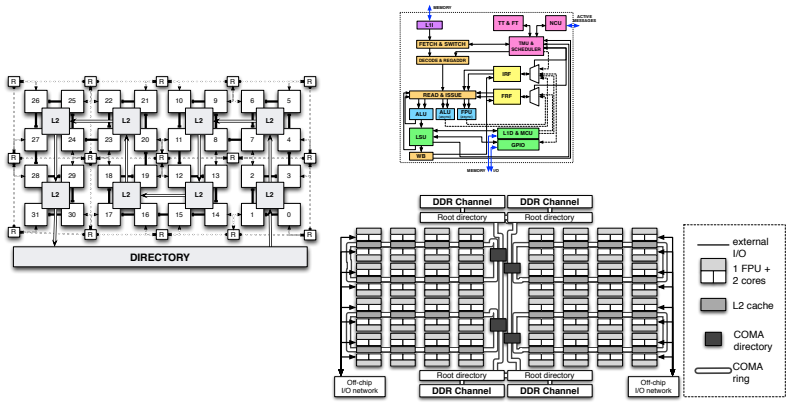
MGSim framework

C++; modular; extensible
for discrete event-driven,
component-based
processor chip simulations

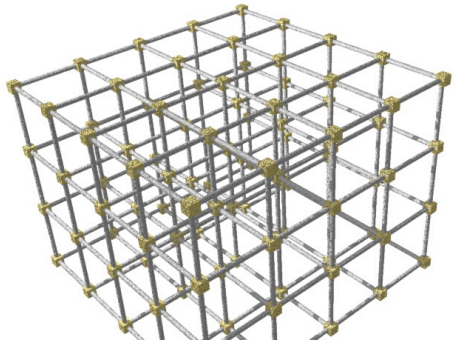


MGSim framework

C++; modular; extensible
for discrete event-driven,
component-based
processor chip simulations

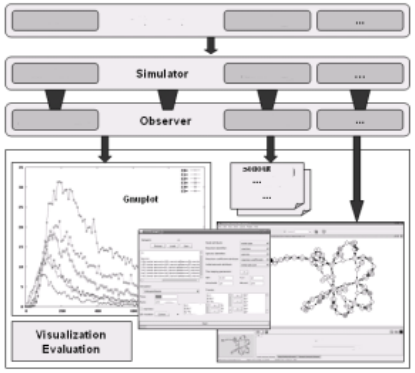


Simulation models for Microgrid components



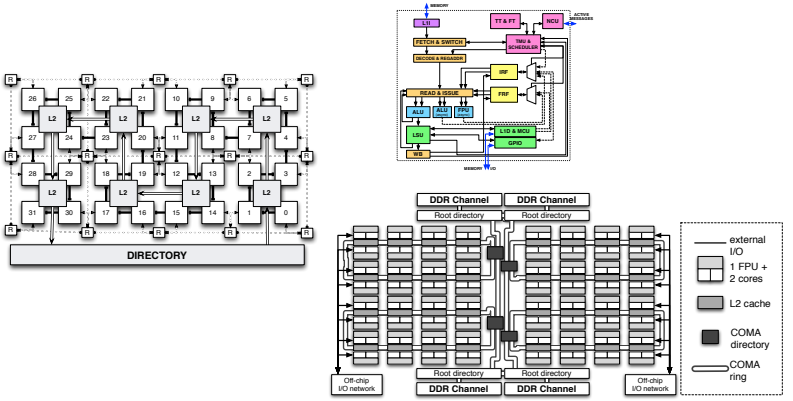
MGSim framework

C++; modular; extensible
for discrete event-driven,
component-based
processor chip simulations

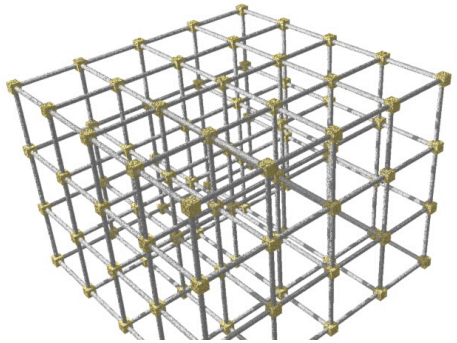


HLSim framework

C++; for discrete event-driven,
simulations of Microgrids

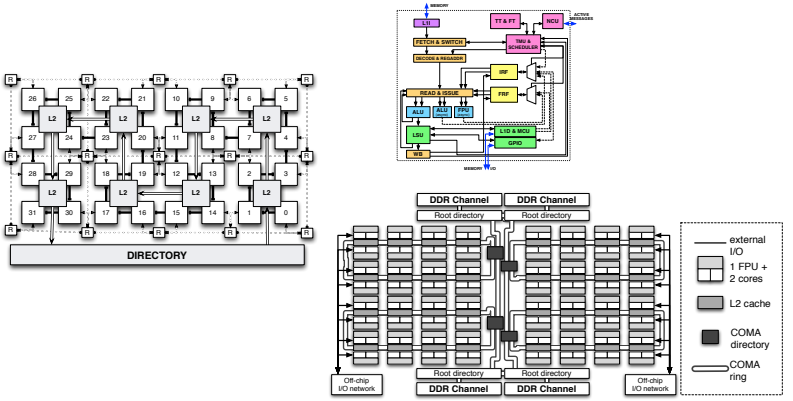


Simulation models for Microgrid components

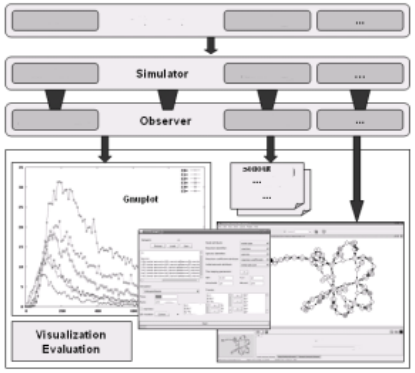


MGSim framework

C++; modular; extensible
for discrete event-driven,
component-based
processor chip simulations

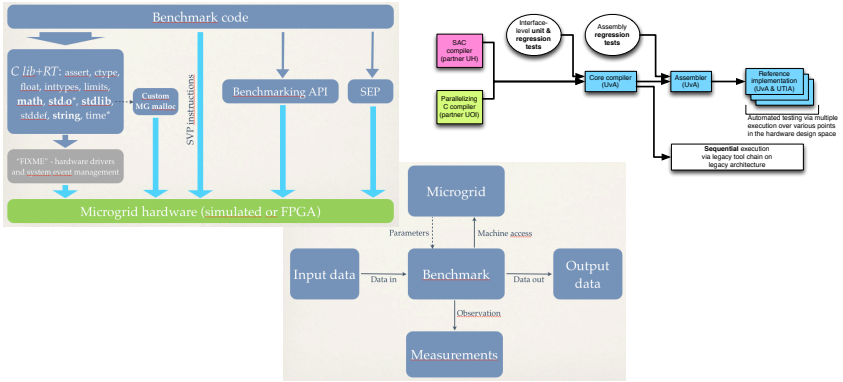


Simulation models for Microgrid components

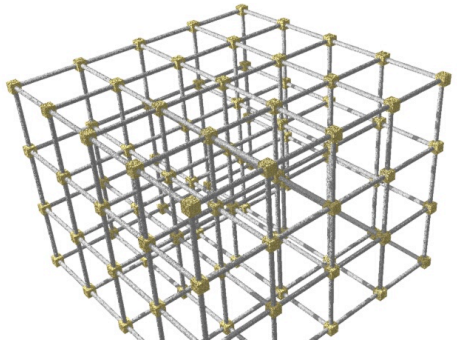


HLSim framework

C++; for discrete event-driven,
simulations of Microgrids

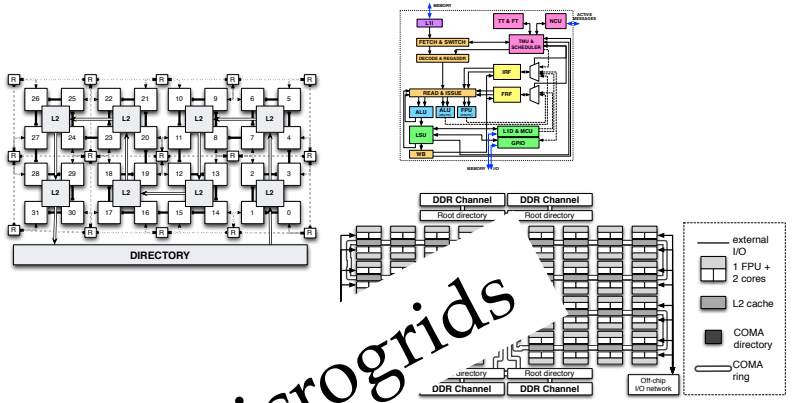


Software tool-chains and benchmarks



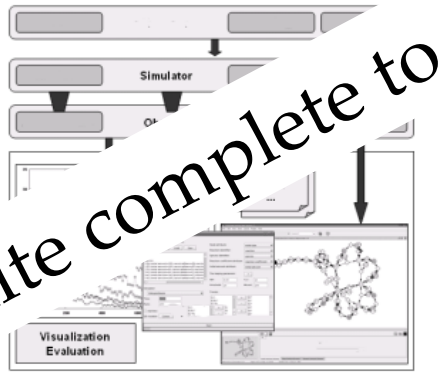
MGSim framework

C++; modular; extensible
for discrete event-driven,
component-based
processor chip simulations



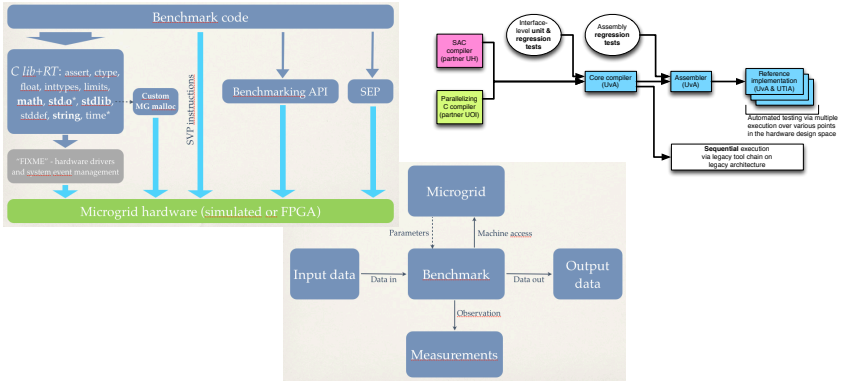
...tion models for ...icrogrid components

Quite complete to continue research on microgrids

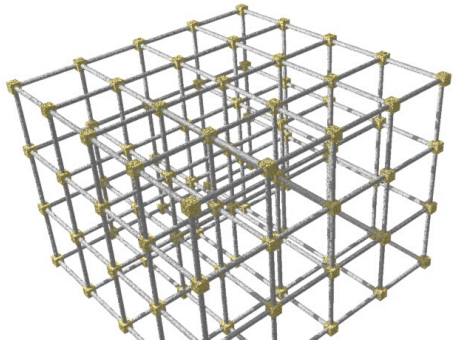


HLSim framework

C++; for discrete event-driven,
simulations of Microgrids

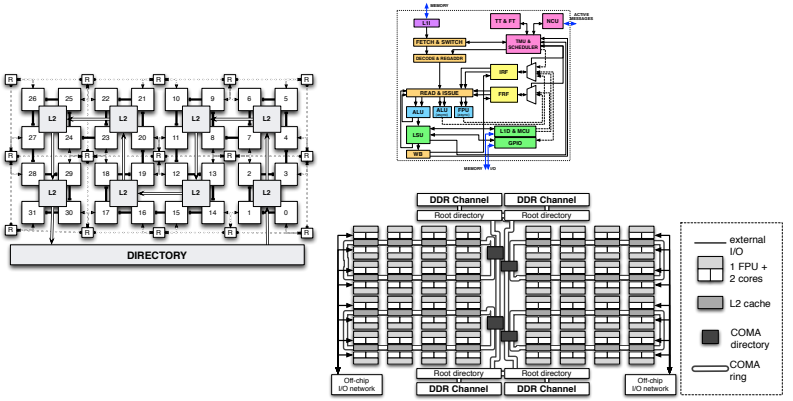


Software tool-chains and benchmarks

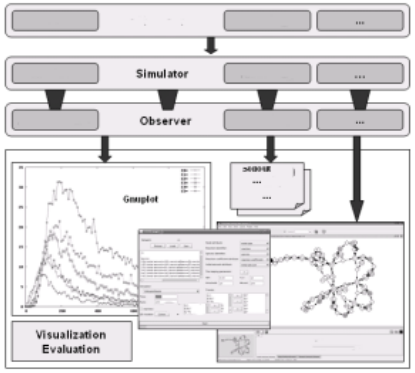


MGSim framework

C++; modular; extensible
for discrete event-driven,
component-based
processor chip simulations

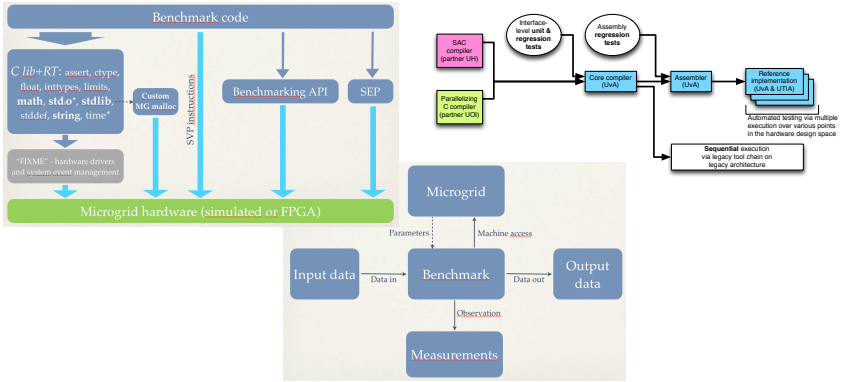


Simulation models for Microgrid components

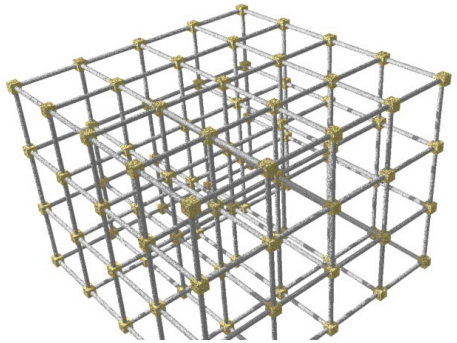


HLSim framework

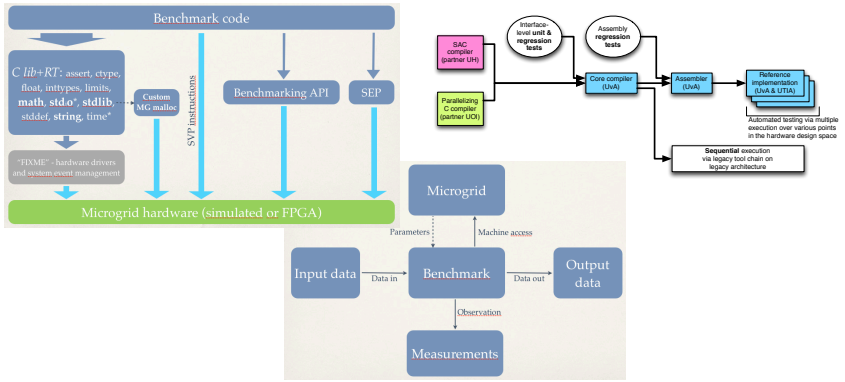
C++; for discrete event-driven,
simulations of Microgrids



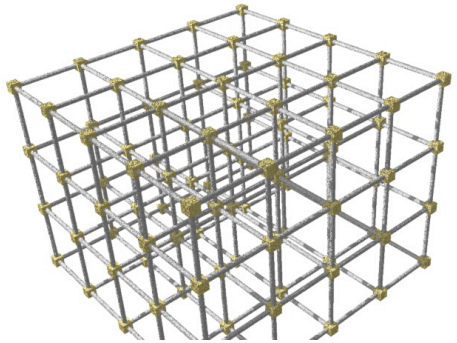
Software tool-chains and benchmarks



MGSim framework

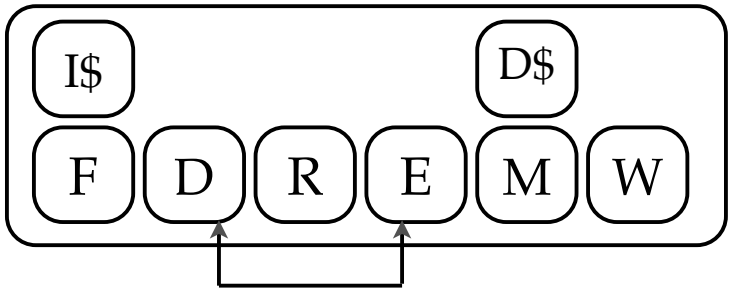


Software tool-chains and benchmarks



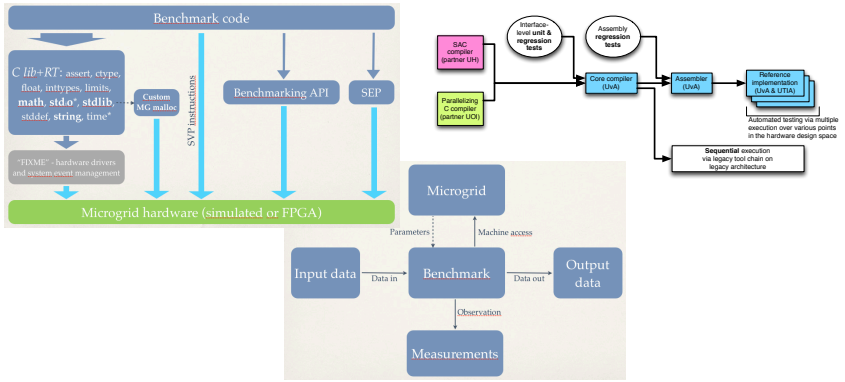
MGSim framework

Simple, configurable RISC model

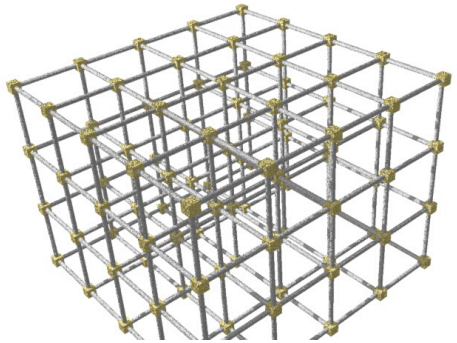


Alpha ISA SPARC

64-bit MIPS 32-bit

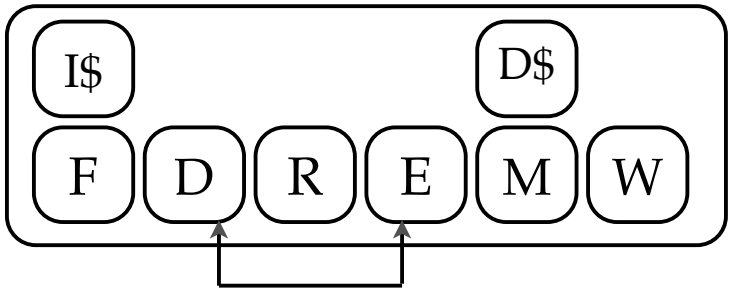


Software tool-chains and benchmarks



MGSim framework

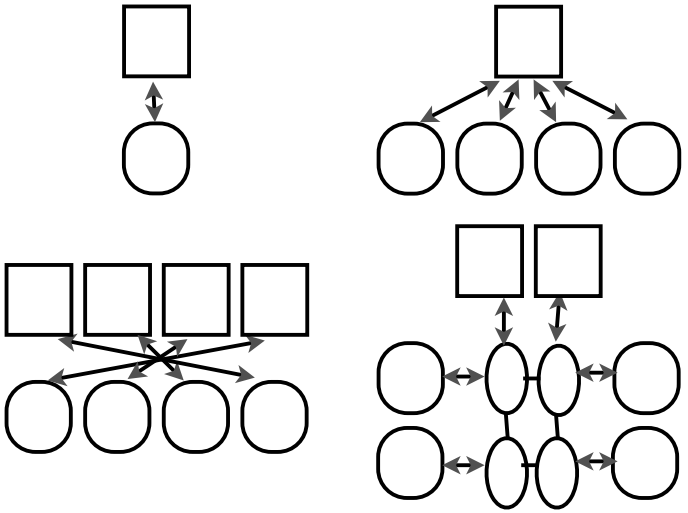
Simple, configurable RISC model



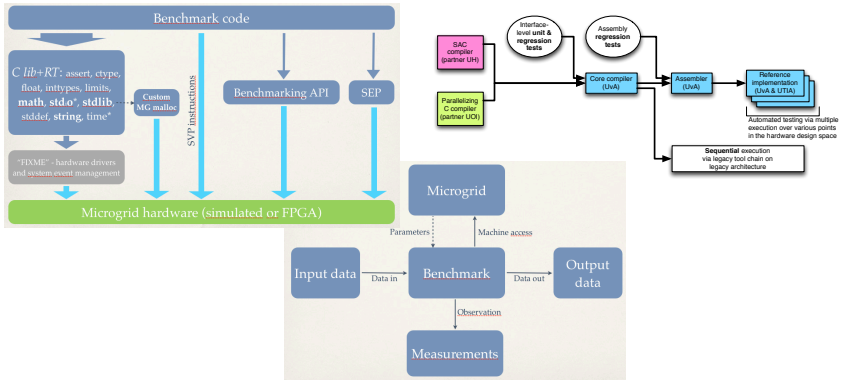
Alpha 64-bit

ISA MIPS

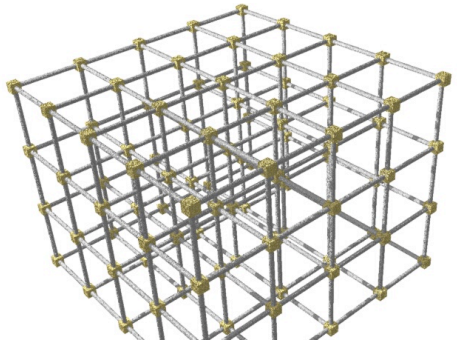
SPARC 32-bit



Configurable memory topology & protocol

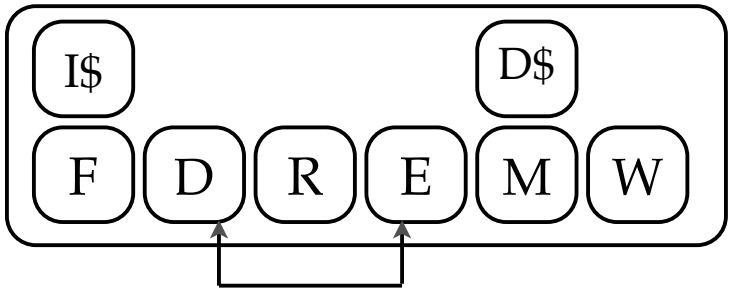


Software tool-chains and benchmarks



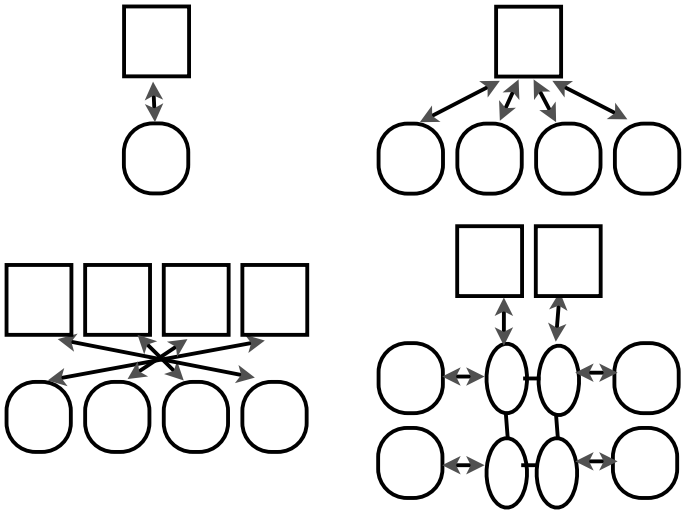
MGSim framework

Simple, configurable RISC model



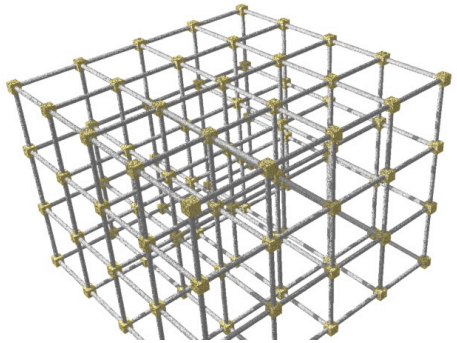
Alpha ISA SPARC

64-bit MIPS 32-bit



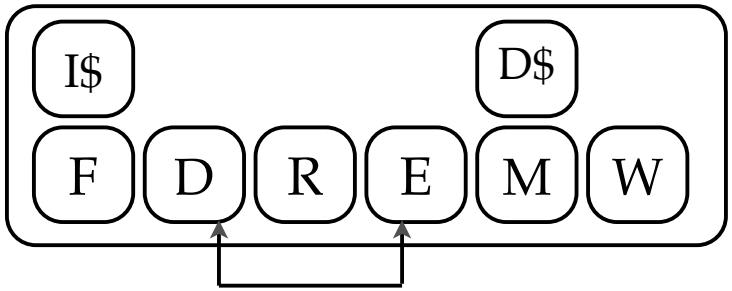
Configurable memory topology & protocol

- C compilers (cross-GCC)
- Operating system components
- Assemblers and linkers (cross-binutils)
- C library components
- Unit tests & functional tests
- Deployment utilities & documentation



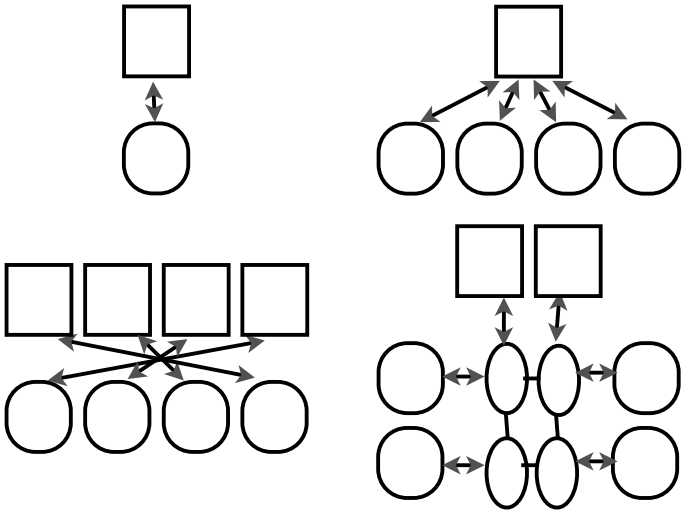
MGSim framework

Simple, configurable RISC model



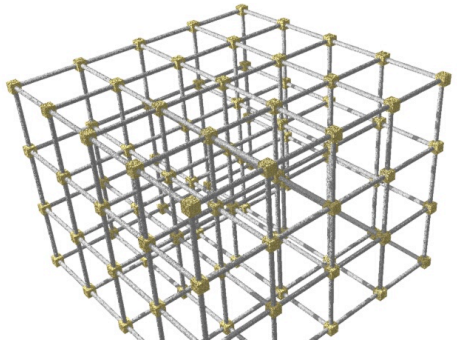
Alpha ISA SPARC

64-bit MIPS 32-bit



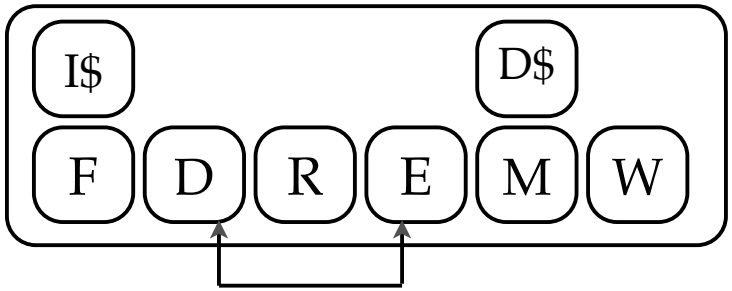
Configurable memory topology & protocol

C compilers (cross-GCC)	Operating system components
Assemblers and linkers (cross-binutils)	C library components
Unit tests & functional tests	Deployment utilities & documentation



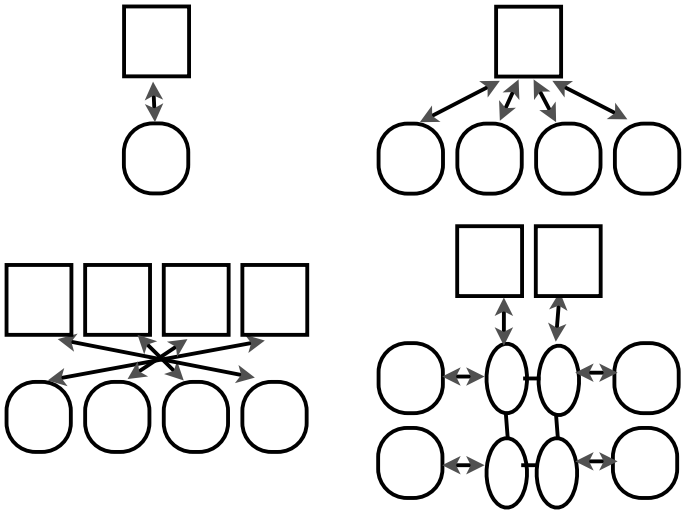
MGSim framework

Simple, configurable RISC model



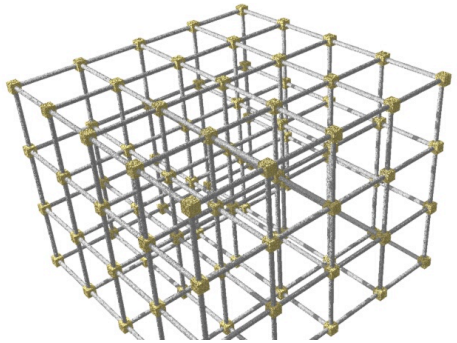
Alpha ISA SPARC

64-bit MIPS 32-bit



Configurable memory topology & protocol

Embedded software tool chain

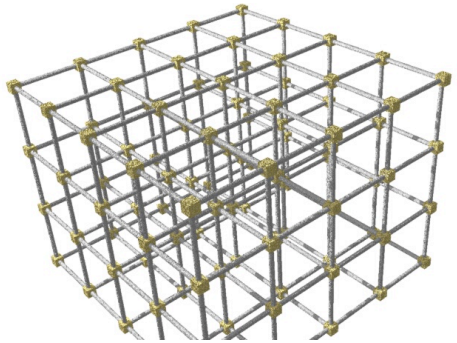


MGSim framework

Simple, configurable RISC model

Configurable memory topology & protocol

Embedded software tool chain

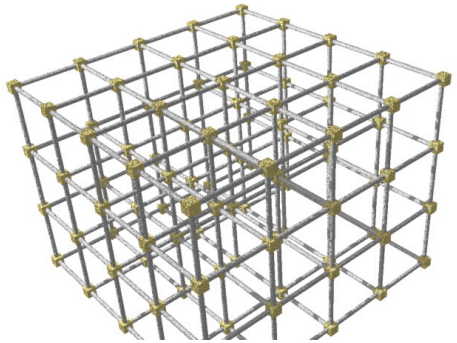


Simple, configurable RISC model

- Sim framework

Configurable memory topology & protocol

Embedded software tool chain

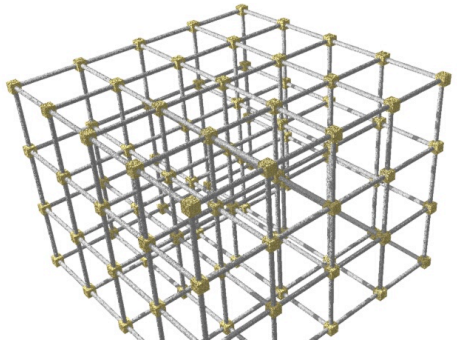


Simple, configurable RISC model

- Sim framework

Configurable memory topology & protocol

Embedded software tool chain



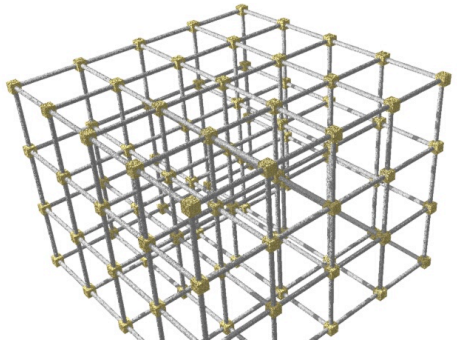
Simple, configurable RISC model

Sim framework

Sexy platform for education & research!
In architecture, code generation & operating systems

Configurable memory topology & protocol

Embedded software tool chain



Simple, configurable RISC model

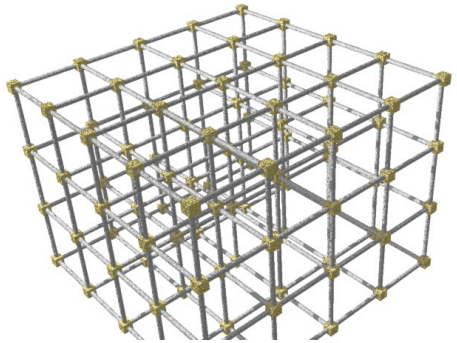
Sim framework

Sexy platform for education & research!
In architecture, code generation & operating systems

simulation traces

Configurable memory topology & protocol

Embedded software tool chain



Simple, configurable RISC model

Sim framework

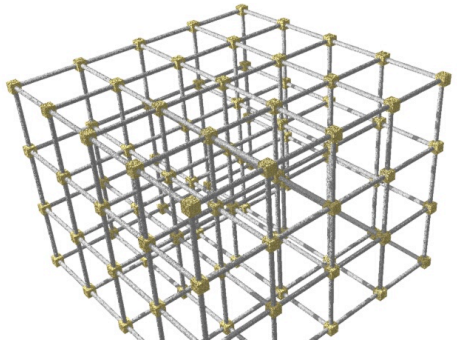
flexible configuration

simulation traces

Sexy platform for education & research!
In architecture, code generation & operating systems

Configurable memory topology & protocol

Embedded software tool chain



Simple, configurable RISC model

Sim framework

flexible configuration

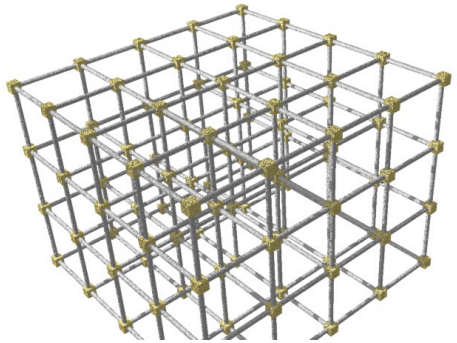
simulation traces

deadlock detection

Sexy platform for education & research!
In architecture, code generation & operating systems

Configurable memory topology & protocol

Embedded software tool chain



Simple, configurable RISC model

Sim framework

flexible configuration

Sexy platform for education & research!
In architecture, code generation & operating systems

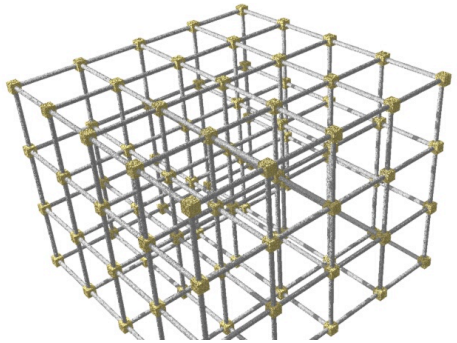
simulation traces

deadlock detection

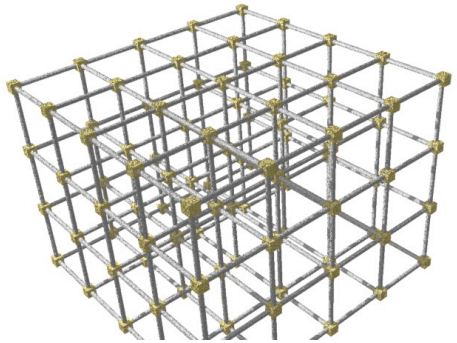
run-time system inspection

Configurable memory topology & protocol

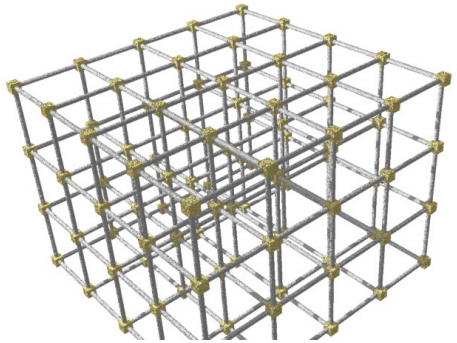
Embedded software tool chain



Sim framework

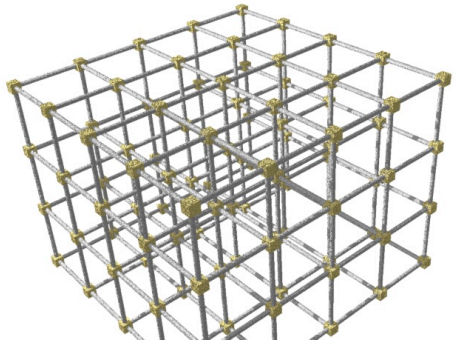


MGSim framework



MGSim framework

Currently: “MicroGrid Simulator”

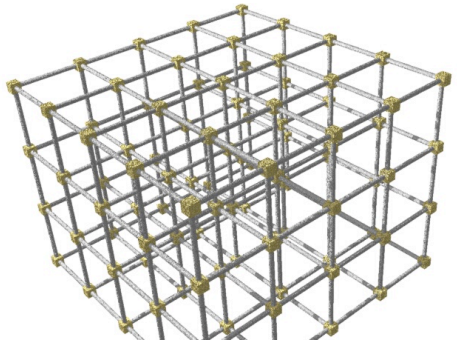


MGSim framework

Currently: “MicroGrid Simulator”

true, but it can do so much more!

(How I feel: “Mike’s Great Simulator”)



MGSim framework

Currently: “MicroGrid Simulator”

true, but it can do so much more!

(How I feel: “Mike’s Great Simulator”)

Proposal: “**Mike’s Groovy Simulator**”

Research paths Using MGSim

Micro-architecture design?

1. Perform research in design
2. Extend MGSim and software infrastructure as needed to run new, custom, parallel programs specialized towards microgrids
3. Write new benchmarks
4. Run benchmarks
5. Get rejected because of lack of baselines...

Possibly: Micro-architecture optimization

1. Extend MGSim and software infrastructure as needed to run existing sequential programs without changes
2. Port / run existing benchmarks to establish baselines
3. Perform research in incremental design
4. Evaluate against baseline
5. Publish!

eg. WCB, fault tolerance etc.

Possibly: Multi-core / memory design space exploration

1. Port / import multi-core or memory-intensive benchmarks
2. Automate simulation over range of core or cache or topology parameters
3. Explore / analyze
4. (Validate against reference platforms)
5. Publish!

In short: Two-faced research stragegy

- **Advertise** MGSim as vehicle to **study existing architectures**
  - May attract researchers working on other architectural problems
  - Will attract code contributions / scrutiny
  - Will stabilize the platform (I/O devices, OS code...)
- **Use internally** MGSim as vehicle to **design new architectural components** (as we already do...)
  - Can then benefit from the reputation/confidence of #1
  - Will capitalize on baselines / applications made possible by #1

Feedback on D-RISC research

- **System features** required to advertise as platform for existing code:
  - Exception handling / Traps
  - Proper address translation (inspiration: SCC)
  - Transparent I/O from any core (inspiration: SCC)
- These features are needed to develop DRISC as well!
  - Will reflect on **virtualizability**
  - Implementation of these features in MGSim must fit the “DRISC vision”; likely to yield **new designs** for exceptions/MMU/IO; these are also publishable!

What about SVP? Microgrids?

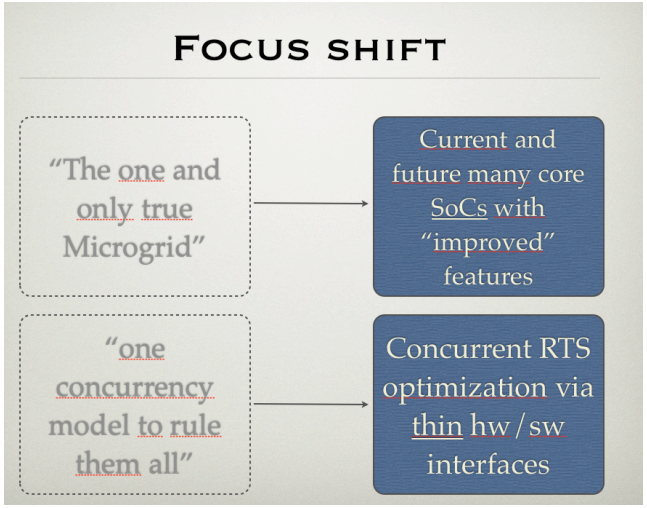
Reminder from Dec. 2011

The new System Virtualization Platform

Motivation / research directions

- **Platform / application co-design**
  to answer the research question:
  "what benefits do we gain from simultaneously evolving both the platform and the program structure towards one another"?
- **Common infrastructure**
  to carry out our overall CSA research activities
  on **both commodity hardware, FPGAs & idealized hardware**
  with a **single set of evaluation applications**

Focus shift



Evolved / remapped concepts

- "the SVP model"
  = *the platform model as perceived by programs that use one of the SVP interfaces*
- "the Microgrid architecture"
  = *an idealized hardware platform for which most SVP interfaces are the thinnest*
- "the SL system language"
  = *our current embedding of the SVP interfaces into the C language*

Updated research question (2012)

- What type of concurrency management intelligence in hardware best supports software OSes to run general-purpose parallel workloads?
  - Answer: probably looks like SVP. (work in progress)
  - Long answer: we can't really know before we evaluate more. There's some new theory involved too.
  - "Generality" still eludes us. Also our audience pushes hard for **cost models** and **virtualizability**!

The message at DSD 2012

- "Microgrids" are **components**
  - containing many simple cores and **hardware intelligence** for concurrency management
- they can be placed as components as **part of larger Systems-on-Chip**
  - including multiple Microgrids side by side
  - including **heterogeneous chips** with legacy cores **connected via the delegation network**

SVP and MGSim: proposed strategy

- "Thin SVP"
  - MGSim + **DRISC + Microgrid protocols**
- "Fat SVP"
  - **Software-managed concurrency** using software scheduler – Implement on MGSim as baseline
- "Alternate SVP"
  - New research: **DRISC + alternate protocols** in hardware to manage concurrency
    – Use MGSim to prototype, use singlethread MGSim baseline applications for evaluation
- Multiple realistic implementations side-by-side will enable more convincing evaluation results.

The sauce to bind them all and make them tasty

- SVP **delegation protocol** and **remote concurrency management**:
  - Define/specify **how components interact**, not what they are made of
  - Use **network-wise compatibility** to define "interoperable software" in an heterogeneous SVP platform
  - Define **cost models** parameterized by SVP resource characteristics
- Very sexy links to "*the computer is the network*" (90's, still popular) and "*everything goes to the cloud*" (2010's, current fad) and "*green computing*" (2010's, likely future hot topic)

When to recognize success

- Able to produce speedup / improvement **diagrams against same-platform baselines across representative benchmarks**
  - If this fails, there is no business with SVP/Microgrids any more!
- Able to write software using common SVP protocol, running on **heterogeneous platforms with respectable parallel performance**
  - If this fails, we must ditch SVP and refocus on DRISC! (or go somewhere else altogether)
- Able to talk about **behavior, performance and operational trade-offs** of concurrent code with an **architecture-agnostic vocabulary / methodology**
  - If this fails, it will still be very good research!

How you can help

Some ideas

- Talk about MGSim as a go-to platform for discovering / studying CA
- Extend/use MGSim to study memory behavior in multi-core SoCs
- Ensure test/benchmark code compiles and runs using the SL tool chain, even if it does not use SL constructs (to ensure code compatibility with the platform)
- Make visualization tools for the data produced during simulations
- Closer integration / interoperability with Gem5?

Thank you!