The RISC Journey from One to a Million Processors

Dave Ditzel
Founder and CTO
Esperanto Technologies, Inc.
dave@esperanto.ai

October 3, 2022
MICRO-55 Keynote
A Personal View of Computer History

Designing computers has been a lifelong passion

Fortunate to have been at the right place a number of times

- Iowa State University: SYMBOL High Level Language Computer & first 8-bit uP chips
- UC Berkeley: At transition from HLLCA to RISC, under prof. D. Patterson
- Bell Labs: At the creation of Unix, C, C++, early RISC processors (C-Machines)
- Sun Microsystems: At transition from M68K to RISC (SPARC)
- Transmeta: Binary translation onto VLIW-RISC
- Intel: Binary translation onto OOO-RISC
- Esperanto: Massively-parallel energy-efficient RISC-V processors

Got to see the evolution of early architectures from 8-bit to 64-bit superscalar monsters

One trend in design seems to have withstood the test of time more than any other: RISC
“Those who cannot remember the past are condemned to repeat it.”

George Santayana,
The Life of Reason,
1905.
Computer Development Timeline of Notable Designs

<table>
<thead>
<tr>
<th>Year</th>
<th>Design/Institute</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
<td>1st Transistor Computer</td>
<td>DEC PDP-1</td>
</tr>
<tr>
<td>1960</td>
<td>DEC VAX</td>
<td>IBM System/360</td>
</tr>
<tr>
<td>1970</td>
<td>Xerox Alto</td>
<td>DEC Alpha</td>
</tr>
<tr>
<td>1980</td>
<td>IBM 801</td>
<td>RISC-V ISA started</td>
</tr>
<tr>
<td>1990</td>
<td>RISC-1,II</td>
<td>RISC-V International</td>
</tr>
<tr>
<td>2000</td>
<td>UCB Stanford MIPS</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>SPARC</td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td>MIPS Co.</td>
<td></td>
</tr>
<tr>
<td>2030</td>
<td>Bell Labs CRISP/Hobbit</td>
<td></td>
</tr>
</tbody>
</table>

CISC Era 20 years

RISC Era approaching 50 years
1978 Photo: My career started at the same time as the first microprocessors

My personal hobby computer

8-bit 6502 Microprocessor

4 MHz

4K Bytes of main memory

Hand “wire-wrapped”

Paper tape reader

Hand wound transformer on power supply.

Note – base of SYMBOL computer in background
Era of
High Level Language Computer Architecture

later called CISC (Complex Instruction-set Computers)
DEC VAX-11/780: a 32-bit CISC

A real 70’s architecture:
- VAX: Transitioned from 16-bit PDP-11 assembly to 32-bit addresses and compiled code
- ISA made it easy to take high level language statements and cast into VAX assembly code

Instruction set heavily influenced by availability of 8-bit wide integrated circuits

VAX ISA composed of variable length instructions
- Instruction opcode (1 or 2 bytes) followed by up to 6 operands
- First operand descriptor (1 byte)
- First operand data (0-4 bytes)
- Second operation descriptor
- Second operand data
- Third operand descriptor
- Third operand data (0-4 bytes)
- ... 

This was great when decoding serially one byte at a time, but a nightmare for pipelined or later superscalar implementations
Xerox Palo Alto Research Center: Alto

First Graphical User Interface with mouse

5.8 MHz microcoded CPU

User loadable instruction sets led to several

HLLCA type bytecoded instruction sets for languages like BCPL, Smalltalk and MESA

Enabled exploration of highly tailored instruction sets

But raised the question

“Why not compile to lowest microcode level?”
SYMBOL: The Ultimate CISC
SYMBOL: The ultimate Complex Instruction Set Computer

SYMBOL was a High Level Language Computer announced in 1971 by Fairchild Semiconductor

Supported by Gordon Moore and Robert Noyce at Fairchild until they left to form Intel

Goal was to reduce cost of software by using hardware instead, literally “programmers cost too much”.

Implemented Compiler, Text Editor and Operating System entirely in logic gates
• 2 gates or 1 Flip-Flop per 14-pin DIP package, no ROM
• 20 thousand chips – took several years to debug at ISU
• Instruction set was bytecode mapped almost 1-1 from high level ALGOL/LISP like typeless language
• Tagged architecture with descriptors, and “logical memory” that was not linearly addressable.
• Five years of my life thinking about CISC vs RISC, i.e. better ways to use 20K chips, programming this machine.

I got to work on this computer while a student for 5 years, after it was donated to ISU

Lessons:
• Don’t build your compiler or OS in logic gates
• Use right combination of Hardware, Software or \(\mu\)Code
Debugging SYMBOL, the inspiration for RISC

Inside red circle are 220 chips.

In order to fix a bug, white wires added on top of blue pc board to make logic changes.

Many of the 100 boards were covered with white bug fixes.

HLLCA Lessons

Putting functions in hardware does not necessarily make them higher performance

Putting functions in hardware always makes them harder to debug, modify and fix

High level opcodes often precluded the ability to make code generation optimizations

Example: Symbol precluded any use of pointer arithmetic, array accesses quite slow.

Better to provide a low level instruction set with HLL Compilers

Use logic gates to make that simple instruction set go very fast

Fascination studying SYMBOL (Ditzel/Patterson) eventually led both to religious conversion to RISC
Mood was changing against HLLCA by end of 70’s

Our “Pre-RISC” paper call for change
ISCA May 1980 (written in 1979)
The Case for the Reduced Instruction Set Computer

David A. Patterson
Computer Science Division
University of California
Berkeley, California 94720

David R. Ditzel
Bell Laboratories
Computing Science Research Center
Murray Hill, New Jersey 07974

INTRODUCTION

One of the primary goals of computer architects is to design computers that are more cost-effective than their predecessors. Cost-effectiveness includes the cost of hardware to manufacture the machine, the cost of programming, and costs incurred related to the architecture in debugging both the initial hardware and subsequent programs. If we review the history of computer families we find that the most common architectural change is the trend toward ever more complex machines. Presumably this additional complexity has a positive tradeoff with regard to the cost-effectiveness of newer models. In this paper we propose that this trend is not always cost-effective, and in fact, may even do more harm than good. We shall examine the case for a Reduced Instruction Set Computer (RISC) being as cost-effective as a Complex Instruction Set Computer (CISC). This paper will argue that the next generation of VLSI computers may be more effectively implemented as RISCs than CISCs.

WORK ON RISC ARCHITECTURES

At Berkeley. Investigation of a RISC architecture has gone on for several months now under the supervision of D.A. Patterson and C.H. Séquin. By a judicious choice of the proper instruction set and the design of a corresponding architecture, we feel that it should be possible to have a very simple instruction set that can be very fast. This may lead to a substantial net gain in overall program execution speed. This is the concept of the Reduced Instruction Set Computer. The implementations of RISC's will almost certainly be less costly than the implementations of CISC's. If we can show that simple architectures are just as effective to the high-level language programmer as CISC's such as VAX or the IBM S/38, we can claim to have made an effective design.

At Bell Labs. A project to design computers based upon measurements of the C programming language has been under investigation by a small number of individuals at Bell Laboratories Computing Science Research Center for a number of years. A prototype 16-bit machine was designed and constructed by A.G. Fraser. 32-bit architectures have been investigated by S.R. Bourne, D.R. Ditzel, and S.C. Johnson. Johnson used an iterative technique of proposing a machine, writing a compiler, measuring the results to propose a better machine, and then repeating the cycle over a dozen times. Though the initial intent was not specifically to come up with a simple design, the result was a RISC-like 32-bit architecture whose code density was as compact as the PDP-11 and VAX [Johnson79].

At IBM. Undoubtedly the best example RISC is the 801 minicomputer, developed by IBM Research in Yorktown Heights, N.Y. [Electronics76] [Datamation79]. This project is several years old and has had a large design team exploring the use of a RISC architecture in combination with very advanced compiler technology. Though many details are lacking their early results seem quite extraordinary. They are able to benchmark programs in a subset of PL/I that runs about five times the performance of an IBM S/370 model 168. We are certainly looking forward to more detailed information.
Why RISC

Trend to “reduce the semantic gap” was the wrong approach.

Turns out a few really simple instructions is a good match for compilers

It’s also much better for the hardware

A simple regular RISC instruction set has simpler control and datapaths and:

- Facilitates efficient instruction pipelining
- Leads to higher operating speed and performance
- Leaves more room for larger caches, for higher performance
- Logic that is “left out” doesn’t have bugs, hence fewer bugs. (lately Musk “the best part is no part”)

But in the early 1980’s RISC vs CISC was not so clear.

- Led to many lively conference debates

We had to build real RISC chips to prove the benefits.
Early RISC Principles

RISC: Reduced Instruction Set Computer
- RISC and CISC terms coined by UC Berkeley professors Dave Patterson and Carlo Sequin

Simple orthogonal instructions
- Often fixed 32-bit length
- Easy for compiler code generation
- Easy to implement with small (10-20) gates per cycle
- Easily pipelined

General purpose register file
- Enough to keep data in registers for current and another called subroutine
- Variation in register file style: Flat, Windowed, or invisible (stack cache)

Single load or store per instruction

Register to register arithmetic operations
IBM 801: In my view, the first true RISC

IBM Introduces the 801 Minicomputer, the First Computer Employing RISC

1974

In 1974 IBM built the first prototype computer employing RISC (Reduced Instruction Set Computer)\(^\text{(1)}\) architecture. Based on an invention by IBM researcher John Cocke\(^\text{(2)}\), the RISC concept simplified the instructions given to run computers, making them faster and more powerful. It was implemented in the experimental IBM 801\(^\text{(3)}\) minicomputer. The goal of the 801 was to execute one instruction per cycle.

In 1987 John Cocke received the A. M. Turing Award for significant contributions in the design and theory of compilers, the architecture of large systems and the development of reduced instruction set computers (RISC); for discovering and systematizing many fundamental transformations now used in optimizing compilers including reduction of operator strength, elimination of common subexpressions, register allocation, constant propagation, and dead code elimination.

The name 801 was from the IBM building number of the T.J. Watson Research Center in Yorktown Heights.

First described at ASPLOS-1 in 1982
Most of die consumed with register windows
More registers to keep more operands on die
Single load or store to/from a register
3-address register to register operations
Integer only
No caches
But simple enough for students to design!
The MIPS instruction set consists of about 111 total instructions

Each instruction encoded in 32-bits

Pipelined to execute one instruction per clock

The instruction set includes:

- 21 arithmetic instructions (+, -, *, /, %)
- 8 logic instructions (&, |, ~)
- 8 bit manipulation instructions
- 12 comparison instructions (>, <, =, >=, <=, ¬)
- 25 branch/jump instructions
- 15 load instructions
- 10 store instructions
- 8 move instructions
- 4 miscellaneous instructions

Delayed branch with 2 delay slots
Early RISC Processors 1978-87: Bell Labs C Machines, including CRISP and Hobbit
AT&T CRISP Microprocessor

C-Language Reduced Instruction Set Processor 1.75 micron CMOS
Announced 1987, 1st CMOS superscalar (more than 1 instr/clock) chip
Hardware translated compact instr into 180-bit wide decoded micro-Op cache
Branch folding eliminated the overhead of branches
“Stack Cache” performed automatic register allocation in hardware
Low power chip, used in EO Tablet computer
Software binary translator used to move software from CISC WE32100

Lessons
• External to internal translation worked well
• Decouple external and internal ISA

Source: D. Ditzel, Hardware Architecture of the CRISP Microprocessor, ISCA, June 1987
AT&T Bell Labs CRISP: C language Reduced Instruction Set Processor
Transition to Bell Labs for 10 years: A more serious look

David R. Ditzel is a member of the technical staff at Bell Laboratories’ Computing Science Research Center in Murray Hill, New Jersey. His current research activities include computer architecture, instruction set analysis, VLSI, computer-aided design tools, and personal computing systems. He has a BS in electrical engineering and a BS in computer science from Iowa State University, where he participated in the Symbol project for four years. In 1979, he received an MS in computer science from the University of California, Berkeley. Ditzel is a member of Tau Beta Pi, Eta Kappa Nu, Phi Beta Kappa, Phi Kappa Phi, ACM, and the IEEE.
Instead of putting our initials onto the chip…..
CRISP: Wins and losses in early tablets

CRISP made it into AT&T EO Tablet in 1993

Did NOT make it into Apple Newton
.... AT&T fumbled the opportunity
and caused Apple to fund ARM

see Wikipedia “AT&T Hobbit” for the story
Lessons in Technical vs Market Success in Hot Chips Presentation (from 2008)

Rank order – My ISA admiration list

1. DEC Alpha
2. MIPS
3. HP-PA
4. Itanium
5. SPARC
6. IBM Power
7. x86

Rank order – Recent Market Success

1. x86
2. IBM Power
3. SPARC
4. Itanium
5. HP-PA
6. MIPS
7. DEC Alpha
Transmeta 1995-2007:

First to provide full x86 compatibility using software binary translation (Code Morphing) to a simpler processor.

Many low power tricks

We proved that doing low power right can be very exciting
Slide that crystallized the concern over power growth

![Power Density Diagram]

Source: Fred Pollack, Intel. New Microprocessor Challenges in the Coming Generations of CMOS Technologies, Micro32

12th Hot Chips Symposium - August 15, 2000
Transmeta low-power x86 Compatible CPU

Efficeon is the sum of

x86 Code Morphing Software

- Provides Compatibility
- Translates the 1’s and 0’s of x86 instructions to equivalent 1’s and 0’s for a simple VLIW processor
- Learns and improves with time

VLIW Hardware

- Very Long Instruction Word processor
- Simple and fast
- Fewer transistors

Low Power

x86 PC Compatibility

High Performance
Bill Gates at Comdex 2000 Keynote announced the Tablet would be the Future of Computing, and held up a Transmeta Crusoe based prototype.

Microsoft’s Tablet PC technology enables any Windows-based application to take advantage of pen-based input.

With software developed by Microsoft, the Tablet PC can function as a sheet of paper.

Handwriting is captured as rich digital ink for immediate or later manipulation, including reformatting and editing.

The Tablet PC requires x86 compatibility as it needs to run Windows XP.
People got really excited about low power Transmeta processors!!
Linus Torvalds led the Transmeta Code Morphing Software team.
Intel 2008-2013
This slide intentionally left blank

\[ x_{\text{86 tax}} = 2x \]
RISC-V

A free and open instruction set
Which instruction set? RISC-V is a better choice

Neither ARM nor x86 is very attractive

- Proprietary and expensive
- Not very energy efficient

New free and open CPU instruction set

- Managed by non-profit RISC-V International
- Over 3000 members
- Simpler ISA, hence more energy efficient
- Allows free or proprietary implementations
  Already competitive in area and performance
  Room for improvement
- Growing ecosystem, like early days of Linux
RISC-V Instructions Described on 2 pages
64-bit ARM Document is ~5000 pages
RISC-V likely to flourish

RISC-V is likely to be the first highly successful open instruction set

Already lots of implementations, some open source, some not

Very successful already for cost sensitive embedded applications

Expect high performance multi-issue out of order versions for high end: SiFive, Ventana, etc

The open nature of the ISA is attractive to many
Present day state of the art:

Esperanto puts over a thousand RISC-V cores on a chip
Esperanto’s approach is different... and we think better for ML Recommendation

Other ML Chip approaches

One Giant Hot Chip uses up power budget
Limited I/O pin budget limits memory BW
Dependence on systolic array multipliers
  • Great for high ResNet50 score
  • Not so good with large sparse memory
Only a handful (10-20) of CPU cores
  • Limited parallelism with CPU cores when problem doesn’t fit onto array multiplier

Standard voltage: Not energy efficient

Esperanto’s better approach

Use multiple low-power chips that still fit within power budget
Performance, pins, memory, bandwidth scale up with more chips

Thousands of general-purpose RISC-V/tensor cores
  • Far more programmable than overly-specialized (eg systolic) hw
  • Thousands of threads help with large sparse memory latency

Full parallelism of thousands of cores always available
Low-voltage operation of transistors is more energy-efficient
  • Lower voltage operation also reduces power
  • Requires both circuit and architecture innovations

Challenge: How to keep the power of each chip to < 20 watts?
Challenge was to put >1000 RISC-V Cores in a 20 Watt chip

Assumed half of 20W power for 1K RISC-V cores, so only 10 mW per core!

\[
\text{Power (Watts)} = C_{\text{dynamic}} \times \text{Voltage}^2 \times \text{Frequency} + \text{Leakage}
\]

<table>
<thead>
<tr>
<th>Power/core</th>
<th>Frequency</th>
<th>Voltage</th>
<th>(C_{\text{dynamic}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic x86 Server core (165W for 24 cores)</td>
<td>7 W</td>
<td>3 GHz</td>
<td>0.850v 2.2nF</td>
</tr>
<tr>
<td>10mW ET-Minion core (~10W for 1K cores)</td>
<td>0.01 W</td>
<td>1 GHz</td>
<td>0.425v 0.04nF</td>
</tr>
<tr>
<td>Reductions needed to hit goals</td>
<td>~700x</td>
<td>3x</td>
<td>4x</td>
</tr>
</tbody>
</table>
Study of energy-efficiency and number of chips to get best ML Performance in 120 watts (six 20W chips)

- 8.5 W with 6 chips
- 20 W with 6 chips
- 118 W with 1 chip
- 164 W with 1 chip
- 275 W with 1 chip

- 2.5x better performance than the 118W chip
- 4x better ML recommendation performance
- 20x better Energy-Efficiency by using lowest voltage instead of highest voltage for our recommendation benchmark

Esperanto's sweet-spot for best performance
Cluster of CPUs (Shires) to Become Next Unit of Compute

For Machine Learning or any highly parallel application wanting to use hundreds to thousands of CPUs, much better to consider the basic unit of compute to be a cluster of CPUs and memory, rather than just isolated individual processors.

Gain dramatic advantages by designing CPUs to work together on large problems: new “Parallel CPU”
- Increase performance
- Reduce area with smaller cores
- Reduce power by enabling simpler design
- Further reduce power (~4x) through cooperation between cores, both data and instruction work

Need to consider memory issues
- L2, L2, L3 cache and path the main memory

Need to consider interconnect
- NoC-to-NoC interconnect between clusters

Esperanto has a highly optimized solution working in silicon today as a great new “unit of compute”
32 ET-Minion CPUs and 4 MB Memory form a “Minion Shire” cluster

32 ET-MINION RISC-V CORES PER MINION SHIRE
Arranged in four 8-core neighborhoods

SOFTWARE CONFIGURABLE MEMORY HIERARCHY
L1 data cache can also be configured as scratchpad
Four 1MB SRAM banks can be partitioned as private L2, shared L3 and scratchpad

SHIRES CONNECTED WITH MESH NETWORK

NEW SYNCHRONIZATION PRIMITIVES
Fast local atomics
Fast local barriers
Fast local credit counter
IPI support
8 ET-Minions form a “Neighborhood”

NEIGHBORHOOD CORES WORK CLOSELY TOGETHER
• Architecture improvements capitalize on physical proximity of 8 cores
• Take advantage that almost always running highly parallel code

OPTIMIZATIONS FROM CORES RUNNING THE SAME CODE
• 8 ET-Minions share single large instruction cache, this is more energy efficient than many separate instruction caches.
• “Cooperative loads” substantially reduce memory traffic to L2 cache

NEW INSTRUCTIONS MAKE COOPERATION MORE EFFICIENT
• New Tensor instructions dramatically cut back on instruction fetch bandwidth
• New instructions for fast local synchronization within group
• New Send-to-Neighbor instructions
• New Receive-from-Neighbor instructions
ET-Minion is an Energy-Efficient RISC-V CPU with a Vector/Tensor Unit

ET-MINION IS A CUSTOM BUILT 64-BIT RISC-V PROCESSOR
- In-order pipeline with low gates/stage to improve MHz at low voltages
- Architecture and circuits optimized to enable low-voltage operation
- Two hardware threads of execution
- Software configurable L1 data-cache and/or scratchpad

ML OPTIMIZED VECTOR/TENSOR UNIT
- 512-bit wide integer per cycle
  - 128 8-bit integer operations per cycle, accumulates to 32-bit Int
  - 256-bit wide floating point per cycle
  - 16 32-bit single precision operations per cycle
  - 32 16-bit half precision operations per cycle
- New multi-cycle Tensor Instructions
  - Can run for up to 512 cycles with one tensor instruction (32K ops)
  - Reduces instruction fetch bandwidth and reduces power
  - RISC-V integer pipeline put to sleep during tensor instructions
- Vector transcendental instructions

OPERATING RANGE: 300 MHz TO 2 GHz

Optimized for energy-efficient ML operations. Each ET-Minion can deliver peak of 128 Int8 GOPS per GHz
Shires are connected to each other and to external memory through Mesh Network.
ET-SoC-1: Full chip internal block diagram

34 MINION SHIRES
• 1088 ET-Minions
8 MEMORY SHIRES
• LPDDR4x DRAM controllers
1 MAXION / IO SHIRE
• 4 ET-Maxions
• 1 RISC-V Service Processor
PCIe SHIRE
160 million bytes of on-die SRAM
x8 PCIe Gen4
Secure Root of Trust
ET-SoC-1 External Chip Interfaces

8-bit PCIe Gen4
- Root/endpoint/both

256-bit wide LPDDR4x
- 4267 MT/s
- 137 GB/s
- ECC support

RISC-V SERVICE PROCESSOR
- Secure Boot
- System Management
- Watchdog timers
- eFuse

EXTERNAL IO
- SMBus
- Serial – I2C/SPI/UART
- GPIO
- FLASH
Summary Statistics and Status of ET-SoC-1

- **ET-SoC-1** is fabricated in TSMC 7nm
  - 24 billion transistors
  - Die-area: 570 mm$^2$

- **1088 ET-Minion energy-efficient 64-bit RISC-V processors**
  - Each with an attached vector/tensor unit
  - Typical operation 300 MHz to 1 GHz

- **4 ET-Maxon 64-bit high-performance RISC-V out-of-order processors**
  - Typical operation 500 MHz to 1.0 GHz

- Over 160 million bytes of on-die SRAM used for caches and scratchpad memory

- **ET-SoC-1 Power ~ 20 watts**, can be adjusted for 10 to 60+ watts under SW control

- **Package**: 45x45mm with 2494 balls to PCB, over 30,000 bumps to die

- **Status**
  - First silicon is healthy and has been shipped to paying customers
Esperanto’s ET-SoC-1 PCIe Evaluation Card

**Esperanto PCI Express Accelerator Card:**

- Single ET-SoC-1 chip running at 300 MHz to 1 GHz
- 1088 ET-Minion cores and 4 ET-Maxion cores
- 1056 ET-Minion cores provide acceleration
- 8 lanes of PCIe Gen 4
- ET-SoC-1 power can be configured from 10W to 60W per chip depending on customer requirements
- 16GB or 32GB DRAM
- Typically used as accelerator to x86 host
- Now being sold in servers for customer evaluations
Esperanto Minion Shire Power with ML Recommendation Benchmarks

Power in Watts for 33 Minion Shires (1056 RISC-V cores)

- A thousand ET-Minion RISC-V processors: 6 watts
- 128 MB of on-die SRAM for caches: 2.8 watts
- 44 Network-on-Chip connections: 1.8 watts

10.6 Watts for 1056 RISC-V processors and L1, L2, L3 caches and Network-on-Chip averages 10.0 mW per RISC-V core
Today: Over 300,000 RISC-V Vector Processors in a single rack

This server has 2 x86 host CPUs and holds 16 Esperanto PCIe cards, each card with one ET-SoC-1 chip with 1088 RISC-V processors, for a total of 17,408 RISC-V processors.

Put 20 servers in a single rack, and that would be 348,160 RISC-V processors.

320 Esperanto PCIe cards in a standard 42U high rack using 20 server chassis, leaving 2U for TOR switch.
What’s next for RISC?

Chiplets and a lot more cores
Standardized Chiplet bus is a big deal.

Will accelerate Chiplet development.

Chiplets will affect your future, so get ready.
Esperanto’s Next Generation will use a Chiplet-based implementation

Easy for Esperanto to make chiplet products, cut into pieces and shrink from 7nm to 3nm
Cost Benefit Using Chiplets Gets More Compelling Every Foundry Node

Example: Relative cost of ONE monolithic 500 mm$^2$ die vs implementing as FOUR 125mm$^2$ chiplets

![Graph showing cost benefit over time for monolithic and chiplet integration.

- **Monolithic (1 Chip, 22.2mmx22.2mm)**
- **Chiplet Integration (4 Chiplets, each 11.1mmx11.1mm)**

- **Same design, but on 4 chips vs 1 chip**
- **Same total chip area**
- **4 chips includes extra testing and assembly cost**
- **Chiplet gains could be more when some pieces done in older less expensive nodes.**
What might chiplets in a package look like in 2030? 4x today in a future node?

Small
1 Host CPU chiplet
1 IO Chiplet
1 Compute Chiplet with:
• 4K RISC-V+vector cores
• 3D Memory

Medium
2 Host CPU chiplet
2 IO Chiplet
4 Compute Chiplets with:
• 16K RISC-V compute cores
• 3D Memory

Large
4 Host CPU chiplets
3 IO Chiplet
16 Compute Chiplets with:
• 64K RISC-V compute cores
• 3D Memory
• Optical IO
Prediction: Over 1,000,000 cores in a small server by 2030

Today in 7nm we have over 1,000 cores per chip
Moore’s law is slowing down, but not dead
By 2030 expect to have over 4,000 cores in a chiplet
Expect to put 16 chiplets together in a single package, ie 64,000 cores in a package
Put each package on a single accelerator board
Like we do today, put 16 of these boards in a small sever

That is over one million RISC-V cores, each with a vector unit, ready to run your workload

What will you do with over a million cores at your disposal. This is the next generation challenge.
One million cores might fit on just a few cards. Challenge is using them efficiently.

Example: 1 Million cores on 16 boards, each board with one package with 64K cores. If boards fit in similar size server as today, that’s 20 million RISC-V cores in a rack.
Energy cost to move data is the key challenge

Today’s ET-SoC-1 vector units can consume ~100TB/s, caches feed them at 8 TB/s. If future chips are 4x as capable, how do we feed 4x, ie 32 TB/second as data set sizes grow?

<table>
<thead>
<tr>
<th>Type of connection to feed RISC-V/Vector cores</th>
<th>pJ/bit</th>
<th>TB/sec</th>
<th>Power Watts</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR5</td>
<td>40</td>
<td>32</td>
<td>10240.0</td>
</tr>
<tr>
<td>PCIe Gen 5</td>
<td>6.5</td>
<td>32</td>
<td>1664.0</td>
</tr>
<tr>
<td>HBM2e. (xfer only, energy to access a bit is 4pJ/bit)</td>
<td>1.8</td>
<td>32</td>
<td>460.8</td>
</tr>
<tr>
<td>Infinity Fabric (AMD)</td>
<td>1.5</td>
<td>32</td>
<td>384.0</td>
</tr>
<tr>
<td>NVLink-C2C (NVIDIA)</td>
<td>1.3</td>
<td>32</td>
<td>332.8</td>
</tr>
<tr>
<td>LPDDR4x (xfer only, energy to access a bit is about 7 pJ/bit)</td>
<td>1.0</td>
<td>32</td>
<td>256.0</td>
</tr>
<tr>
<td>SAINTS-S (Samsung)</td>
<td>0.80</td>
<td>32</td>
<td>204.8</td>
</tr>
<tr>
<td>CoWoS (TSMC)</td>
<td>0.56</td>
<td>32</td>
<td>143.4</td>
</tr>
<tr>
<td>Bunch of Wires</td>
<td>0.50</td>
<td>32</td>
<td>128.0</td>
</tr>
<tr>
<td>EMI (Intel)</td>
<td>0.30</td>
<td>32</td>
<td>76.8</td>
</tr>
<tr>
<td>UCle long distance</td>
<td>0.50</td>
<td>32</td>
<td>128.0</td>
</tr>
<tr>
<td>UCle short distance</td>
<td>0.25</td>
<td>32</td>
<td>64.0</td>
</tr>
<tr>
<td>3D via TSV</td>
<td>0.20</td>
<td>32</td>
<td>51.2</td>
</tr>
<tr>
<td>On-die long distance</td>
<td>0.10</td>
<td>32</td>
<td>25.6</td>
</tr>
<tr>
<td>On-die short distance</td>
<td>0.01</td>
<td>32</td>
<td>2.6</td>
</tr>
</tbody>
</table>

Data movement power may be far greater than compute power. Seems like stacking memory in 3D on top of compute chiplet may be a way to reduce excessive chip to chip data movement power.
Summary

RISC vs CISC is no longer a popular debate, RISC has won

• RISC techniques are ubiquitous in most types of processors
• RISC-V is likely to be the common instruction set for much new RISC innovation
• RISC-V doesn’t carry years of baggage, providing advantages in power, area, cost
• RISC-V started with simple embedded applications
• Will soon see 8+ issue RISC-V out-of-order cores rivaling the best of other ISA implementations
• For massively parallel compute, energy efficiency is key, and slower in-order with vector wins

Our future is likely to continue to use RISC techniques:

• Easy to program general-purpose RISC-V likely to win over specialized ISA & architectures
• Next big challenge likely to be methods to program millions of RISC-V cores efficiently
• Big opportunities in CPU micro-architecture to design more efficient clusters of RISC-V cores
Thanks!

End of Presentation

Questions?