Singularity
Design Note
30

Multi-core Singularity (MCS)

Current hardware trends are set to double the number of CPU cores on die every 12-18 months for at least the next 5 years. Current generation of I/O devices contain specialized cores that are not exposed to the operating system and applications for general purpose use. The meaningful utilization of these processing cores is presently an open question. This document opens the discussion on how Singularity could be architected for the multi-core systems.

1. Introduction 3

2. Background 3

2.1. Commodity MP Systems in 2006 3

2.2. Multi-core motivators 5

2.3. Multi-core performance 5

3. Current Hardware Datapoints 6

3.1.1. Many core processors 6

3.1.2. Dedicated core processors and offload processors 6

3.1.3. Mixed core systems 7

3.1.4. FPGA systems 7

4. A Multi-core Singularity 8

5. Additional Opportunities 8

5.1. Hardware support for Channels 8

5.2. Dedicating cores to kernels and applications 8

5.3. Core aware scheduling 9

5.4. Leveraging heterogeneous processors9

5.5. Hiding heterogeneity 9

5.6. Concurrent programming and transactional memory 9

6. TODO Error! Bookmark not defined.

References 9

7. History Error! Bookmark not defined.

1. Introduction

A proliferation of cores exists today within a typical PC (CPU, GPU, disk controller, soundcard). Commodity CPUs are shipping with dual cores and some with hardware thread support. The current industry targets are to double the number of cores every 18 months. As we go forward we may see heterogeneous cores on each die – low power core for low intensity applications and higher power cores for more intensive applications. This document proposes a design for a multi-core Singularity that leverages the system's current strengths and does not impinge on efforts elsewhere in the company.

For the purposes of this document, we consider a core to be a complete CPU or a hardware thread on a CPU, and regard a multi-core system to have between 8 and 1024 cores. The problems with multi-core systems are facing all OS vendors. There is a wide range of existing research in areas relating to the construction of these systems: concurrency, programming models, system design.

The question naturally arises as to how we support these cores and how may of them might we reasonably use them on the Singularity platform. There are issues in programming models and system design that need to be addressed, as well as potential Singularity can provide a good vehicle for research in this space because:

- Singularity has a small code base and is amenable to change.
- It has modular kernel and runtime systems that could be tailored to support specific types of processing core.
- Singularity uses linked stacks and is potentially able to support more threads than systems that use fixed stack sizes.
- The software isolated processes and channel communication provide an efficient message passing infrastructure.
- The usage MSIL as an intermediate language decouples the system from ISA dependencies to a reasonable degree.

There is a multi-core task force within Microsoft that exists to identify and address issues relating to multi-core. As multi-core systems become more prevalent, there will be increased pressure to address the issues in Windows. The Singularity project's multi-core work should play to the strengths of the Singularity platform and avoid duplicating ongoing efforts with Windows.

The outline of this document is as follows: Section 2 provides background on how multi-core systems have reached the commodity market; Section 3 presents some interesting hardware developments relating to multi-core; Section 4 presents a straw man multi-core architecture for Singularity; and Section 5 identifies research opportunities based on the Singularity platform.

2. Background

This section presents an overview of the design of commodity MP systems today and leads into a discussion on why multi-core systems are the future.

2.1. Commodity MP Systems in 2006

There are presently two types of multiprocessor systems shipping as commodity hardware. The processors in a s*ymmetric multiprocessor* system (SMP) share common I/O and memory buses. The processors have equal access to resources and are likely to be in contention for memory bandwidth with current hardware clock rates. In *non-uniform memory architecture* systems (NUMA) the processors are arranged into nodes that have their own local memory. The nodes may access the memory in other nodes, but this is slower because of the need to forward the request across multiple buses. The I/O buses in a NUMA system are connected to particular nodes and I/O interrupts are routed to the nodes on the bus rather than interrupting all processors.

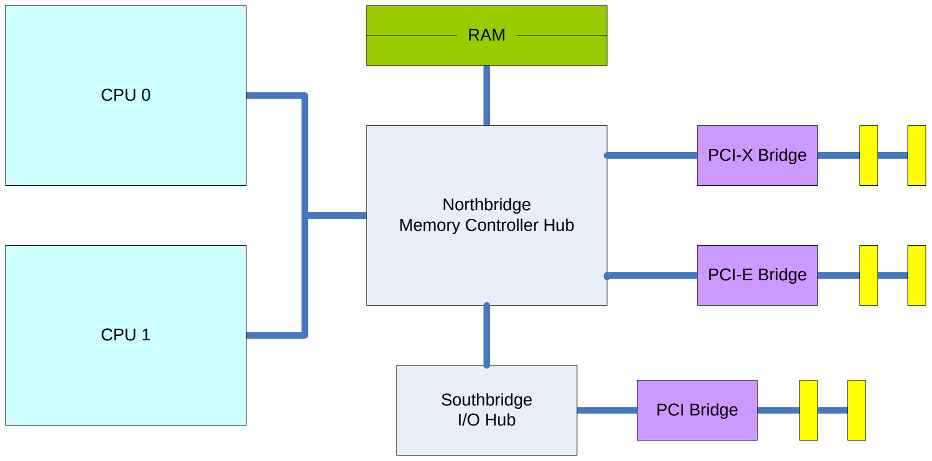

Figure 1: Symmetric Multiprocessor PC architecture

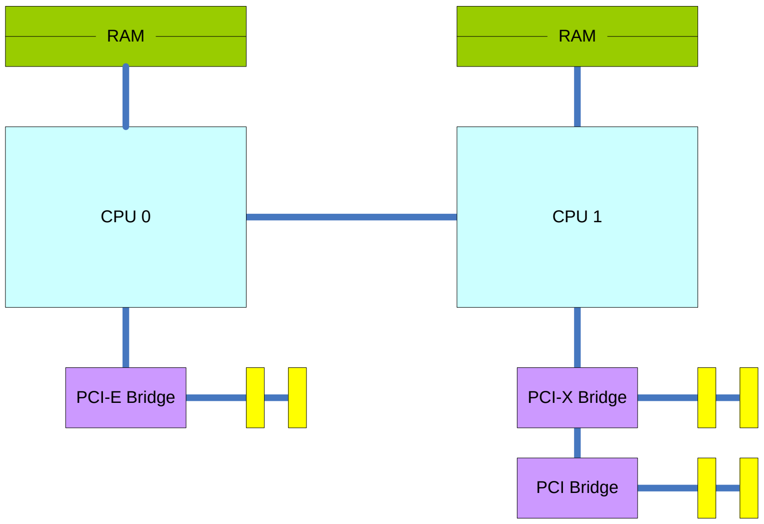

Figure 2: Two node NUMA PC architecture

Small scale MP machines based on Intel's PC architecture are SMP machines. These machines share the Northbridge connection to the off-processor memory controller and I/O buses. An example of this design is shown in Figure 1. The design places pressure on the bandwidth limited Northbridge and obviates good design elsewhere. A pertinent example being the point-to-point I/O architecture of PCI-Express - a design to maximize I/O performance by not sharing I/O links. Intel will undoubtedly address the shortcomings of the Northbridge in the near future.

AMD based MP systems are NUMA designs. AMD integrated the memory controller on the die of its 64-bit processors and developed a fast inter-processor and I/O bus called HyperTransport. HyperTransport can provide up to 11 GB/s unidirectional bandwidth and supports 3 billion bus transfers per second. These systems have ample I/O bandwidth for most purposes. An example of this design is shown in Figure 2. AMDs processor product lines are differentiated by the number of HyperTransport links each product line supports. The Athlon64 product line is intended for home systems and supports 1 link, whereas the Opteron server lines support one, two, and three hypertransport links for 1-, 2-, and 4-way NUMA systems. The processor dies are shipped with one or two cores and each processor die is a NUMA node. A number of vendors, such as Newisys [Kota05], provide HyperTransport hardware that facilitates building systems with larger collections of nodes using toroidal interconnects.

In both SMP and NUMA systems, the processors run cache-coherency protocols to ensure each processor has a consistent view of memory through the caches. There are multiple models of cache-coherency. The x86 model is defined as being [TODO]…perils of cache-line sharing.

2.2. Multi-core motivators

There are two issues driving the development of multi-core systems: the first is the processor speed wall, and the second is the memory wall [wulf95]. Processor manufacturers have found it increasingly difficult to improve processor clock rates. They are limited by physical processes. However, Moore's law advances on the number of transistors on a silicon die has continued and processor manufacturers have more transistors per unit cost available to deploy. The present answer for Intel and AMD is to place more processor cores on each die[1] to produce chip multi-processors (CMP). On The cores on each die share the processors physical connections. In Intel's case the processor is attached to the front front-side bus, and in AMD's case the memory subsystem and hyper-transport links.

For the period before processors hit the speed wall, processor speeds and cache speeds largely increased in the lock step and double with alarming regularity. However, during this period memory latency and bandwidth enjoyed more conservative increases. The relative cost of cache misses grew and the mismatch became known as the memory wall. Modern processors may stall for tens or hundreds of cycles whilst waiting on main memory, and find their rate of computation limited by the memory bottleneck. The response to the memory wall is hardware threading. The processor supports multiple thread contexts and switches between them to mask memory latency. In *fine-grained multithreading*, the threads are switched every clock cycle. In *coarse-grained multithreading*, threads are switched on operations that will stall the processing, such as cache misses. On super-scalar architectures, *simultaneous multithreading*, refers to the opportunistic issue of instructions from multiple threads within a clock cycle when the functional units involved would otherwise be idle[2].

The common way to expose hardware threads to the operating system is to report hardware threads as it they were separate processors. An MP capable operating system is able to support hardware threads without modification, though there are performance advantages if the OS knows that the threads share caches.

2.3. Multi-core performance

Stretch, local vs remote memory in NUMA.

Number of applications on W32 benefiting from MP.

[1] The pinnacle of Intel's manufacturing technology in 2006 is the IA64 Montecito with 1.9 billion transistors. The Yonah core manufactured for laptops in 2006 contains just 151 million transistors.

[2] Intel's hyper-threading technology is an example of simulataneous multi-threading.

3. Current Hardware Datapoints

The research community has 40 years experience building distributed and multi-processor systems. A review of this space is beyond the scope of this document. Instead, we focus on a few current trends that may provide an insight into how hardware is going to develop over the next 10 years.

3.1.1. Many core processors

Support for hardware threads has been added to many of the major processor families (Alpha, ARM, PowerPC, Sparc, x86). Sun Microsystem's Niagara architecture [Kong05] is the most ambitious process that is shipping in volume. It features 8 processors with 4 coarse grained hardware threads per processor and is shown in Figure 3.

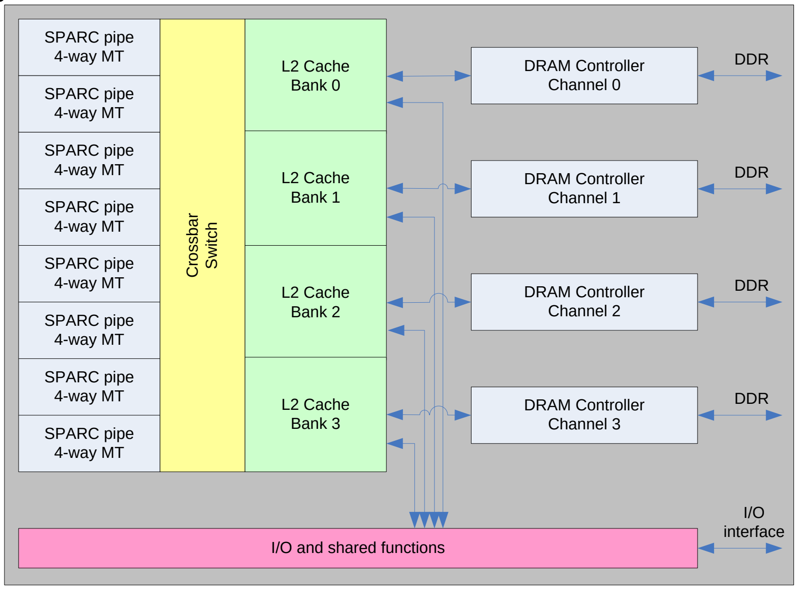

Figure 3: Architecture of Sun Microsystems Niagara Processor (adapted from [Kong05]).

The Niagara design also features split level 2 caches (3MB total) and four separate memory controllers to reduce contention. The Niagara processor shares a floating point unit between all of the cores. Thus the processor is good at throughput tasks, but not strong with computational tasks.

The Cray Multi-threaded Architecture (Tera MTA) [Alverson90,Carter99] features 128 hardware threads on a processor and has no data caches at all. Each thread may have up to 8 memory references in flight. The Tera MTA also uses full-empty bits to support programming with futures – a thread may block waiting for a value to be written to an empty memory address and be scheduled by the hardware when the value is written. The Cray MTA also has 256 processors arranged in a 16x16x16 toroidal mesh and has dedicated I/O processors feeding the computation processors. The system is a long way from where commodity machines are today, but shows the scale hardware threads can produce. In addition, the richness of I/O controllers on commodity hardware is increasing and could potentially be leveraged by Singularity.

3.1.2. Dedicated core processors and offload processors

There is a long history of processor offload units on-die and off-die. These include floating point co-processors, SIMD vector units (Sparc VMX,Intel MMX, AMD 3D now, PowerPC Altivec/VMX), and crypto-graphic accelerators (VIA C7). The goal is increase system performance along an axis deemed to attractive. Interaction with the offload processors has to be co-ordinated and the offload units may be supported by programming library changes or by being made to look like a separate hardware device.

The Sony, Toshiba, IBM Cell processor is an interesting data point on the horizon – the Cell processor is to ship in Sony's Playstation 3 and IBM has prototype workstations and blade servers. The Cell processor features a conventional PowerPC CPU and 7 or 8 synergistic processing elements (SPEs) that are RISC cores with SIMD instructions to accelerate media workloads. SPE memory access restricted to

a dual ported regions shared by CPU. The main CPU is responsible for scheduling instruction streams and data for the SPE's. The Cell processor is advertised as having high peak floating point performance values for (256 SP GFLOPs and 30 DP GFLOPs) [Wang]. There is no associated programming break-through for working with Cell and the complexity of the SPEs may make it hard to wring performance out of them.
GPUs + accelerator
Another point on the horizon is the ClearSpeed CSX600 [CSX600], which is a 96 core dedicated floating point processor with a PCI-Express interface. It is intended to be installed on commodity PCs and is programmed using modified versions of standard mathematics libraries. The cores are relatively simple The claimed sustained throughput per processor is 25 DP GFLOPs whilst consuming just 5W of power. Development boards include multiple processors. The processors on these systems are VLIW processors with an ALU and FPU and 6K of scratchpad memory. Systems filled out with these boards are competing for the crown of world's most powerful supercomputer. It's feasible that similar units may be integrated on to processor dies as the CPU vendors compete with the GPU makers.
Alessandro Forin et al wrote a technical report on using a dedicated vector processor on the Talisman multimedia card [Forin98]. The card had an ARM processor for running programs and the vector processor was used for MPEG decoding. It provides details on how they co-coordinated task scheduling, which requires delicate handling because the vector processor has a large amount of state.
These devices and others like them are interesting because of their raw processing power. Singularity could support offload principles, but there are arguably no architectural principles to apply to facilitate this.

3.1.3. Mixed core systems

Kumar et al [Kumar03] examine the performance of a multiprocessor system comprising of 4 processors from a compatible instruction set (Alpha EV-4, EV-5, EV-6, EV-8) and examine the potential for saving power by switching core. They claim a 39% power reduction for a 3% decrease in performance whilst running SPEC benchmarks. The approach could offer large energy savings in a wide range of scenarios and might be taken up by the major vendors as they become more interested in performance-per-watt than outright performance.

Balakrisnan et al [Bala05] examine a multi-core system with processors operating at different clock rates. Their system is comprised of 4 processors which are each configured to run at one of two speeds. They further wrote a processor affinity API to bind threads to processors. They then ran a wide range of multithreaded applications and tasks and looked for instability in runtimes. Their findings are largely intuitive:

- With a mix of fast and slow processors, processing performance is reduced to that of the slowest processors when threads share memory.
- An O/S scheduler aware of asymmetry helps for certain types of application (SPECjbb, Apache). In other cases the application needs to be explicitly aware to avoid instability (DB2/TPC-H, SPEC OMP).
- Having mixed performance cores results in better than expected mid-point performance compared to all slow and all fast cores.

The results of these papers suggest paths the processor vendors could take to get better power utilization and reduce manufacturing complexity. An implementation in Singularity that switched processor clock rates would be relatively straightforward and it would be a good feature to have for server farms.

3.1.4. FPGA systems

There is a group at the University of Kansas [Andrews04] looking at developing symbiotic hardware and software compilers. Applications are compiled into software for a traditional processor with logic to generate gates on an FPGA to run some of the application threads in hardware. It's an interesting idea.
Berkeley RAMP [RAMP] project is looking at synthesizing many cores on a small number of FPGA boxes. They are looking at constructing systems with 64 to 1024 cores.
Alessandro Forin's work on add custom instruction support on FPGA systems to improve performance for particular applications.

4. A Multi-core Singularity

The current high end of the consumer MP market is held by NUMA architecture machines. This is the likely direction both of the major vendors will pursue because of its scaling properties. The obvious question is whether Singularity should become a NUMA capable OS. Operating systems supporting NUMA typically support a range of page table techniques to try to minimize the number of remote memory accesses and share memory load between nodes for shared data. Adding this support would require a substantial engineering effort and since Windows already has a solid NUMA implementation, we would do well to focus our efforts in another direction.

An arrangement that would allow Singularity to run on NUMA hardware and leverage Singularity's strengths would be to treat the nodes in a NUMA system as largely independent processing engines. Each node would run its own kernel instance and take responsibility for scheduling its own threads and managing its own memory. We would provide the necessary infrastructure to route channel communications between processes running on separate nodes and design it to maximize the node performance.

Conceptually the system would resemble a Transputer system except the individual processors are replaced by multiple cores. Our distinguishing feature would be the use of contracts on the inter-node communications.

The system will face similar issues to NUMA systems, such how to place processes for effective I/O data paths and how to migrate processes between nodes. However, the complexity required in the kernel is substantially less because of the limited memory address space interactions. We will also have the ability to customize the nodes in the system independently of each other. One node could run a soft real-time kernel and own all of the real-time processes in the system for example.

XXX Diagram XXX

We will also have to determine whether all the nodes in the system need to run all of the kernel. In particular, the security service and namespace, stand out as features that might want to run on all nodes for reliability against node failures, or just on one node to keep the code simpler.

The proposed system is akin to distributed system in a box and whilst building distributed systems out of Singularity has been outside the projects goals, it may allow Singularity to scale across clusters and integrate with mobile devices. This will require input on revisions to the security mode, the namespace, and how the system deals with failures.

5. Additional Opportunities

5.1. Hardware support for Channels

Chuck Thacker has suggested the idea of hardware support for channels and this would greatly simplify IPC with many cores. Today we can copy data into well known locations and send an inter-processor interrupt, but with hardware support we would just send the message. With or without hardware support the issue exists of when to interrupt the receiving processor, i.e. immediately, or when the threads on the sending processor are all blocked, or at some other point.

5.2. Dedicating cores to kernels and applications

Once the number of cores gets large we might consider an alternative design treating the system as having little or no shared memory to reduce contention and running distinct instances of the O/S on nodes. This would be conceptually similar to the explicit scheduler control that applications had on Transputer systems, but provide richer functionality. It would be feasible to construct real-time kernel and application runtimes to support the execution of real-time applications, like videoconferencing applications, on sets of dedicated cores. The kernels associated with cores could be statically or dynamically adjusted to support the application mix. Depending on the interconnection fabric and processor to bus connections there may need to be higher level with I/O scheduling for high bandwidth applications, like live HDTV video compression.

In a similar spirit, the system might dedicate cores to supporting legacy and alternative operating systems and applications.

5.3. Core aware scheduling

Cores distributed within a die and on separate dies will have different cache and memory localities. There has been some research on cache aware scheduling [Fedor2005] and we could push in this direction with greater variation in the caches.

5.4. Leveraging heterogeneous processors

Singularity might have some traction in utilizing mixed ISA processors on a system. For instance, the majority of modern disk controller contain a reasonably powerful micro-controller (Seagate recently cited an ARM and 32MB of RAM). Assuming we can target ARM in the foreseeable future, we could look at running I/O related tasks on these systems and low load tasks. We can emulate the type of access needed with development boards and turning off the functionality natively provided by the device; in the case of a disk controller this would mean disabling caching and implementing our own scheduling and caching algorithm along with our own special sauce such as encryption. In the case of a network processor, we could run the IP stack instances on network cards.

What are compelling applications?

For a disk controller, we could run the disk driver process on the controller, or we could run low level filter services such as RAID or crypto, or even the whole file system service instances on the controller.

For a network controller, we could run packet filtering on the card to avoid generating interrupts to the main processors for firewalled traffic. Or, we could distribute the network stack across network controllers.

For the GPU, we can treat it as a very fast SP floating point unit. And we might find some way to utilize the memory. Is there any benefit to running general purpose code? Do we have some compelling enough apps to merit porting Accelerator?

There is a reasonably long history of I/O processors, but I'm not yet aware of ones running user applications or O/S services.

5.5. Hiding heterogeneity

Singularity could support custom processing elements, such as the floating point pipelines of the GPU, through custom runtimes or with custom code generators. Applications built in this way would need their manifests to present the hardware requirements to the O/S.

5.6. Concurrent programming and transactional memory

Not sure of any distinctions here between multi-core systems and conventional MP systems. This is a wide open space.

References

[Alverson90] Robert Alverson, David Callahan, Daniel Cummings, Brian Koblenz, Allan Porterfield, and Burton Smith, “The Tera Computer System”, ACM International Conference on Supercomputing, pp. 1-6, June 1990.
[Andrews04] David Andrews, Douglas Niehaus, Razali Jidin, Michael Finley, Wesley Peck, Michael Frisbie, Jorge Ortiz, Ed Komp, and Peter Ashenden, “Programming Models for Hybrid FPGA-CPU Computational Components: A Missing Link”, IEEE Micro, July-August 2004.http://wiki.ittc.ku.edu/hybridthread/images/6/69/Micro.pdf
[Azul] Azul Systems, Compute Appliance for Network Attached Processing, http://www.azulsystems.com/products/cpools_cappliance.html
[Bala05], Saisanthosh Balakrishnan, Ravi Rajwar, Mike Upton, Konrad Lai, “The Impact of Performance Asymmetry in Emerging Multicore Architectures”, 32nd Annual International Symposium on Computer Architecture (ISCA'05), pp. 506-517, 2005.
[Carter99] Larry Carter, John Feo, and Allan Snaveley, “Performance and Programming Experience on the Tera MTA”, SIAM Conference on Parallel Processing, March 1999.
[CSX600] ClearSpeed Technology, “CSX600 datasheet”, http://www.clearspeed.com/downloads/CSX600Processor.pdf

[Fedor05] Alexandra Fedorova, Margo Seltzer, Michael D. Smith, and Christopher Small, “CASC: A Cache-Aware Scheduling Algorithm For Multithreaded Chip Processors”, Sun Microsystems, Technical Report, March 2005. http://research.sun.com/scalable/pubs/CASC.pdf
[Forin98] Alessandro Forin, Andrew Raffman, and Jerry Van Aken, “Asymmetric Real Time Scheduling on a Multimedia Processor”, Microsoft Technical Report, MSR-TR-98-09, February 1998. Available from: http://research.microsoft.com/research/pubs/view.aspx?type=Technical%20Report&id=134
[Glass05] Adam Glass and Peter Johnston, “NUMA Specification for .NET”, May 2005.
[Kong05] Poonacha Kongetira, Kathigamar Aingaran, Kunle Olukotun, “Niagara: A 32-Way Multithreaded Sparc Processor”, IEEE Micro, March-April 2005, pp21-29.
[Kota05] Rajesh Kota and Rich Oehler, “Horus: Large Scale Symmetric Multiprocessing for Opteron Systems”, IEEE Micro, March-April 2005, pp30-40.
[Kumar03] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Pathasarathy Ranganathan, and Dean Tullsen, “Single-ISA Heterogenous Multi-Core Architectures: The Potential for Processor Power Reduction”, Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36), 2003.
[RAMP] http://ramp.eecs.berkeley.edu/about.php
[Wang] David Wang, “Report from ISSCC 2005 on the Cell Processor”, http://www.realworldtech.com/page.cfm?ArticleID=RWT021005084318&p=1
[wulf95] Wm. A. Wulf and Sally A. McKee, “Hitting the Memory Wall: Implications of the Obvious”, Computer Architecture News, Volume 23, Number 1, pages 20—24, 1995.
[XXX] Missing reference to company making multi-core MIPS processors. I believe they had 16 or 24 MIPS64 cores on 1 die, but can no longer find the reference.