The Case for a Cray on a Chip

P Machanick[0000-0001-6648-7032]

Rhodes University, Makhanda, South Africa
p.machanick@ru.ac.za

Abstract. Moore's Law is usually interpreted as a prediction of how many transistors you can buy for the same money at some future date. It can also be interpreted as how long you need to wait until a given number of transistors falls below a target price. An example of this reverse-application of Moore's Law is transitions such as the emergence of microprocessors competitive with traditional larger-scale computers and the emergence of smartphones. Since the late 1990s, it has become increasingly common for growth in transistors to equate to more CPUs (cores) per die. Recent designs have over 50-billion transistors and far more potential parallelism than can be supported by memory. I argue the case for a rebalancing of design goals with a much larger, faster on-chip memory and a CPU that is designed around this memory system. The proposal: a Cray-class vector CPU on a die with 1 Gibyte of static RAM, or Crayon (for Cray on a chip). The kind of organization classically used by Cray vector supercomputers is feasible to achieve on a single chip. I argue that a design like this can use the available memory bandwidth, as opposed to over-CPU designs with a large number of cores and GPU threads that are memory limited and propose how such a design could be used.

Keywords: supercomputer vector architecture Moore's Law.

1 Introduction

The purpose of this paper is to predict a new technology inflection point, based on the same logic that could have been used to predict timing of when single-chip microprocessors would challenge the dominance of the mainframe and when it became feasible to build a smartphone at an affordable price. The specific prediction is that a vector CPU along the lines of an early Cray design, with a substantial on-chip static RAM memory, will soon be possible. Making the prediction presages more detailed studies to explore the design space.

Why is such a development a good idea? Memory access is a significant bottleneck in existing designs, so a design study reversing the trend of cramming more and more CPU power onto a chip is worth considering. Instead, the amount of CPU should be balanced with a significant-sized fast on-chip memory.

This paper makes the case for such a design alternative, while leaving open details, which can only be established with a full design study including simulations. Such a design study will need suitable workloads and to consider variations such as the ratio

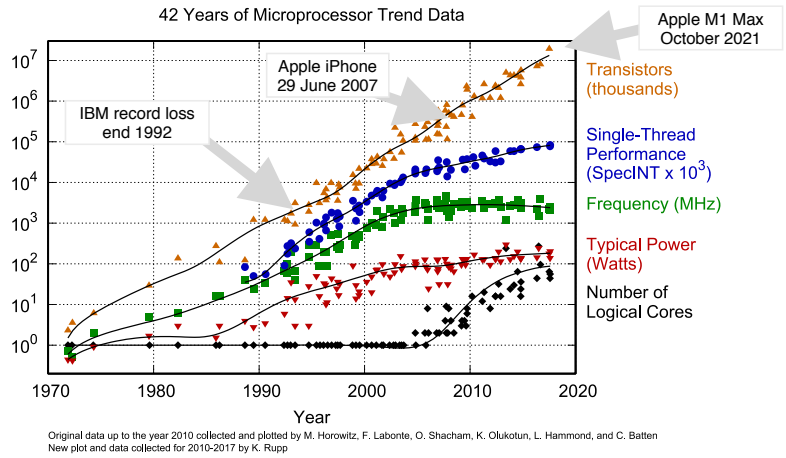


Fig. 1. Moore's Law trend showing major inflection points. When IBM made their record loss, RISC designs such as MIPS R4000 were appearing and Intel's Pentium was about to launch. The iPhone required a good enough ARM CPU. Apple M1 Max has 57-billion transistors.

of transistors used for RAM versus logic and implementation of a high-speed network to scale up to large-scale systems.

Fig. 1 shows the general Moore's Law trend [15], with a few major inflection points added. In 1992, IBM had the then biggest loss in US corporate history [23]. That coincided with the rise of the "killer micro": cheap microprocessors that could supplant the high-performance computing market for mainframes at a fraction of the price. Only a few years later, specialist supercomputer makers started filing for bankruptcy [37]. The second inflection point is mid-2007, when it was possible to buy enough hardware to run a Unix-class kernel and still have enough resources for a decent application layer, making smartphones possible (some preceded the iPhone, but the iPhone launch marks the turning point where smartphones started to become ubiquitous). The third inflection point is late 2021, when processors with more than 50-billion transistors on a die started to appear.

What is the significance of these inflection points?

The classic view of Moore's Law is that it predicts how much more you can buy for the same money as a function of elapsed time, typically twice the transistors every eighteen months; repeated claims of the demise of this trend are another constant [44]. The transistor count trend line illustrated by the graphed examples hints at that. This is a useful metric, as it predicts how long a current purchase will last before it becomes obsolete. It also predicts how expensive a purchase is worth making now if you cannot use all its performance or expansion capacity immediately. If you spend down now, you may be able to do better next year.

The inflection points hint at another trend line: instead of a constant-price line, a constant-functionality line predicts when a certain level of functionality will fall through a price point of interest. A killer micro became possible when a single-chip

Table 1. Terminology.

Term		Definition
Ki	kibi-	binary prefix for $2^{10} = 1024$
Mi	mebi-	binary prefix for $2^{20} = 1024^2$
Gi	gibi-	binary prefix for $2^{30} = 1024^3$
Ti	tebi-	binary prefix for $2^{40} = 1024^4$
die		chip (as seen by hardware designers)
core		CPU or GPU on a die; if more than one
DRAM		dynamic random access memory
		each bit is a capacitor and must be refreshed
SRAM		static random access memory
		each bit is made of transistors and is not refreshed
ROM		read-only memory
bank		separately accessible unit of memory
ISA	instruction-set architecture	programmer-visible instructions and registers
microcode		low level instructions used to implement ISA
control store		ROM used to store microcode
μ-ops	micro-operations	simplified instructions generated in hardware
cache		fast SRAM memory close to CPU
		fastest is L1, nearest CPU, then L2, etc.
issue		instruction moves to execute pipeline stage
multi-issue		more than one instruction can execute at once
scalar		CPU that executes one instruction at a time
superscalar		CPU with a multi-issue pipeline

microprocessor with full functionality (integer, floating point and memory manager) could be made. A smartphone became possible when enough transistors could be bought, within the budget for a phone CPU, to run a good kernel and nontrivial graphical applications.

Gordon Bell predicts new classes of computer should emerge every 10 years [4] based on a similar analysis – though his categories differ as he does not consider devices like smartphones.

In Table 1, I summarize terminology to aid the non-specialist. Though I prefer using binary prefixes for memory sizes as those accurately reflect the hardware, if I cite a source that uses decimal prefixes like M and G (e.g., in Section 2), I use their terms, even if they may be inaccurate.

Having reached this point: should we not be looking for a new inflection point, based on delivering something previously impossible, now that over 50-billion transistors can be put on a die? In the remainder of this paper, I review the limitations of cramming CPU power onto a die, touch on trends in addition to Moore's Law and look for historical designs that could better use so many transistors. I select one and outline why it is a good idea, and where it takes us next. I end with conclusions.

2 Limits of Over-CPU Designs

It is useful to backtrack a bit to how we got to the current situation, where processor-heavy designs increasingly dominate the industry. In 1996, Kunle Olukutan made the case for a single-chip multiprocessor, now commonly referred to as multicore, based on the observation that aggressive multi-issue pipelines have diminishing returns. 4 CPUs able to issue two instructions per clock need about the same chip space as a single CPU able to issue 6 instructions per clock. Despite 3 times the peak throughput, the more aggressive design at best is 30% faster than the simpler design on a single thread or process and 4 simpler CPUs do better on multithreaded or multitasking workloads [30].

At the time Olukutan was making the case for a single-chip multiprocessor, one of the more aggressive designs available, the AMD-L6, had 88-million transistors and could issue 6 instructions – either real instructions, or micro-operations (μ-ops)[1] derived from real instructions – in one clock cycle [16]. Apple's M1 Max with 57-billion transistors has nearly 650 times as many. What are all these extra transistors used for? Some details have been made public [27] though Apple does not disclose as much as some. M1 Max has 10 regular CPUs, 32 GPUs and 16 NPUs (neural processing units, claimed to be capable of 11 trillion operations per second – though it is not clear if this is in the aggregate or per NPU). To make this all work without being totally memory-limited, Apple packages DRAM tightly with the CPU in a System in a Package (SiP) design. The 512-bit memory bus has four 16-bit 128-bit LPDDR5 channels. Cores are split into performance and efficiency; the former share a 12 MByte L2 cache and the latter a 4 MB L2 cache. Performance cores each have 192 KB of instruction and 128KB of data cache, and efficiency cores respectively 128 KB and 64 KB. With 8 performance and 2 efficiency cores, this adds up to nearly 3 MB of level 1 cache, for a total over all cores and levels of nearly 20 MB of cache.

Not taking into account the cache-control logic, with 6 transistors per bit (the usual SRAM implementation [45] though other variations are possible), this amount of cache requires about 950-million transistors, or less than 2% of the total of 57-billion transistors.

A design with as much CPU on one die as Apple's M1 Max is memory-constrained. Though Apple sells its designs with tightly-integrated DRAM, satisfying demand for so much processing is challenging. If all 58 processing units (10 CPUs, 32 GPUs, 16

[1] As a way of bridging the gap between complex instruction sets and RISC, some designers use μ-ops that break programmer-visible instructions into simpler operations; μ-ops are designed to be easier to pipeline than complex instructions [20].

NPUs) try to access DRAM at once, that becomes a major bottleneck. Even with a high bandwidth bus, contention is likely to be an issue particularly as all units are unlikely to be in lockstep and accessing the same region of memory. It would be interesting to know how many memory transactions are required to be interleaved with 11-trillion (1.1×10^{13}) NPU operations per second to sustain that rate with real-world computations. To add to that, for the conventional CPUs, though the L1 caches are a decent size, the L2 caches are not particularly large.

During the original supercomputer era, it was well known that there could be a very big gap between peak and achievable throughput, as revealed by an attempt at standardising application-level benchmarks [10]. Having a massive theoretical peak throughput that is not achievable has to be weighed against other uses of hardware resources that could be effectively utilised.

When the amount of computing power on a die is out of balance with reasonably achievable memory resources, I call this an "over-CPU" design.

A major drawback of Apple's approach to countering an over-CPU design by tightly integrating DRAM into the package is that it removes the option of memory upgrades. For a consumer computer, this is problematic as a buyer on a limited budget can stretch the usability of a machine by buying more DRAM, particularly as DRAM prices drop. Worse, it is not a sustainable strategy for future designs as the latency gain of this sort of packaging can only happen once; if the long-term trend of a growing speed gap between CPU and DRAM persists, an over-CPU strategy will see diminishing returns.

The Apple M1 Max exemplifies the problem; it is not the only example. Recent NVIDIA GPU designs have 80-billion transistors, with 50 Mbytes of cache [13], accounting for about 3% of the total transistors.

3 Trends and Learning Curves

There is a long history of changes in optimum instruction set architecture (ISA) being driven by memory technology. The ISA in broad terms is the programmer-visible instruction set, including instructions and registers. In very early systems with small memories, designs that led to compact code were optimal [12]. When a fast read-only memory (ROM) became an option, a *control store* containing *microcode* – code in a very low-level form that interpreted the ISA layer – became viable. Microcode liberated ISA designers from simplicity: as bigger control stores became possible, increasingly complex ISAs became feasible.

As memory became cheaper and memory footprint of code became less of an issue, designs favouring a simplified hardware design without microcode and facilitating aggressive pipelines – even if at the cost of needing more memory – became a more viable option. The Reduced Instruction Set Computer (RISC) [34] movement consequently arose. Also feeding into this was that semiconductor memories replaced magnetic core memories, so it was no longer possible to make a small control store that was many times faster than main memory and control store was no faster than a cache [33].

The RISC movement was to some extent inspired by Seymour Cray's designs while at Control Data. The CDC 6600 is credited with being the first supercomputer as well

as presaging RISC ISA principles [2]. Cray's later designs when he started his own company exploited the invention of semiconductor memory to the full. Aside from adding vector instructions and registers, the Cray-1 used what was then a large static RAM (SRAM) main memory, divided into 16 banks[2] [41]. In the remainder of this paper, when I refer to a "Cray" architecture, I mean a classic vector design, not any later designs that may have appeared under the Cray name. Cray in this sense has become a descriptive term rather than a name of a specific product.

Moore's Law is a specific example of a *learning curve law*: essentially a law of competition. If all competitors expect an increase of N% per year on some metric, they all aim for that. If they aim too high, they risk their new design being ready too late. If they aim too low, they risk losing market share to the leaders.

In general terms, if C(q) is the metric of interest for making the qth item in a series of improvements and C(1) is the measure of the first, for the parameter of learning p, Equation 1 defines a learning curve [50].

$$C(q)=C(1)^{-p} \qquad (1)$$

Moore's Law is the most famous example but trends in dynamic RAM (DRAM) are equally important. While density drives speed of logic, in DRAM density mainly drives capacity. Consequently, there is a growing CPU-DRAM speed gap, long predicted to run into the *memory wall*, where CPU speed improvements will be masked by memory access delay [48]. Added to this, DRAM organization is becoming increasingly complex, with timing of refresh yet another issue to take into account for minimum latency [5]. The memory wall was to some extent evaded by the shift to multi-core designs with lower clock speeds but the underlying problem is still there and an over-CPU design does not help with balancing DRAM latency with CPU speed.

Across the industry there are other learning curves of interest such as improved power management, improved battery life and reduced network switching latency. However, for this paper, memory and CPU trends are sufficient to illustrate the point: an over-CPU design is not a good use of available transistors.

Another trend is the growing general-purpose use of GPUs (GPGPU). This approach is opportunistic: GPUs offer a lot of computational power and are relatively inexpensive compared with a custom high-performance CPU. However, they are widely known to be difficult to program [11]. I have argued elsewhere that we will reach a GPU endpoint: a faster GPU will not be useful up to the point where human senses are saturated. Once that point is reached, speed enhancements of GPUs will only benefit general-purpose use. I further argue that once this point is reached, a sub-optimal GPU that is easier to program for general-purpose use may be a better design trade off [25]. While Intel explored part of that design space with the abandoned Larabee project, that CPU was based on a modest pipeline with added graphics instructions, rather than a reasonably strong computation engine [43]. The better solution in my view is to start with a known high-performance ISA and, if necessary, extend it with graphics operations.

---

[2] Large, in its day, meant 64 Mibytes.

3.1 Other Related Work

The RISC-V project provides open designs for a RISC architecture drawing on lessons of older designs, a starting point for developing new ideas unencumbered by licensing considerations [2]. There are various extensions of the basic RISC-V ISA, including vector instructions inspired by Cray designs [40].

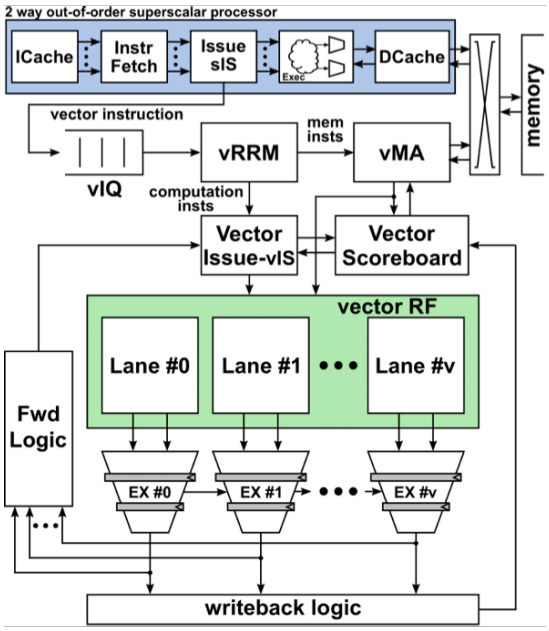


Fig. 2. RISC-V^2 Vector pipeline showing the major components.

A growing number of projects is building on the RISC-V vector architecture [21, 47, 31, 36]. Allied to this is work on improved vector code generation in the LLVM compiler project specifically for the RISC-V vector ISA [1][3].

It is therefore not a sticking point for a project to design a Cray-like vector CPU as the RISC-V free CPU project has already worked on that [31] – see key components of one design, RISC-V^2, in Figure 2. RISC-V^2 includes a 2-way superscalar pipeline with out of order execution, much the same as each single core in the original single-chip multiprocessor proposal. So the major work in the project is designing the memory system and the software layers. Another project has an even more complete RISC-V vector design [36] so it is increasingly becoming a matter of choosing the one closest to requirements as a starting point.

[3] The GCC toolchain is also available but LLVM generally has better performance [38].

Where it comes to balancing CPU and on-chip memory, an idea that has been around for some time is processing in memory (PIM) [18], also sometimes called intelligent RAM (IRAM) [32].

Why has this idea not taken off? A clue is in the Terasys design [18], where relatively modest CPUs were incorporated into DRAM. DRAM and CPU logic are built with different processes and combining the two on one die involves design compromises, making it difficult to get the best possible DRAM or best possible CPU design on the same part. Consequently, recent work on PIM [28] has focused on tight packaging of DRAM dies with CPU dies (such as 3-dimensional stacking – see also Apple's SiP packaging [27] referred to in Section 2), more complex DRAM organization that can mitigate off-chip delays and new alternatives to DRAM, none of which have yet become available at scale.

3.2 Putting it all together

The overall proposal here is to combine free CPU work of the RISC-V project with the PIM idea. Taking advantage of the large number of transistors of recent designs (the "inflection point"), instead of implementing PIM using DRAM, and hence limiting the capability of the CPU, using SRAM allows an aggressive CPU.

Unlike the trend of contemporary designs, the balance is shifted to more relatively high-speed memory on a die, rather than more CPU.

4 A More balanced design option

Given the potential for doing something different and new now that a die with more than 50-billion transistors is viable, should we be looking at more of the same, or looking for a new idea?

The answer, I propose, is reviving an old idea in a more efficiently packaged and hence lower-cost form.

My idea is that a reasonable use of over 50-billion transistors is to use most of them for SRAM on a chip with a single processor close to the design of a Cray vector machine.

Why SRAM and not DRAM?

While DRAM has many modes to improve speed of access [14], it cannot achieve the same flexibility and overall speed as SRAM. Part of the speed difference arises from the different electronics. SRAM uses transistors to store bits. DRAM stores a bit using a capacitor, which needs to be refreshed. Switching time is inherently slower than a transistor. Another issue is that the fabrication technology is different, making it harder to include logic and DRAM on one chip, while SRAM uses the same logic as a CPU.

Such a design would be far simpler than a typical over-CPU architecture. Multiple banks of SRAM on one chip would not require complex logic. A single vector CPU with out of order execution of scalar instructions would be similar to the Cray-1. With a large amount of on-chip SRAM, the CPU would not be memory-limited.

Why Cray? Cray's designs were very competitive in their day, though expensive to build as they used a large number of discrete components and needed very efficient packaging to minimise communication latencies, hence the very compact design particularly of the early models [3]. Vector compilers became very efficient and some of the ideas like multiple banks of SRAM are not particularly difficult to implement on a single chip.

How much SRAM?

A die with over 50-billion transistors could support 1 Gibyte of SRAM with space left over for a CPU. With 6 transistors per bit, 1 Gibyte of SRAM requires 51.5-billion transistors. This is about 90% of the transistor count on an M1 Max die, leaving nearly 10% of a similar-sized die for CPU logic. However, the exact amount of SRAM that would be optimal would depend on how fast it could be accessed. That is a function of wiring delay as well as switching speed. Both of these factors improve as feature size reduces. Apple's M1 Max is built with a 5nm process and 4nm processes are already available[4].

A reasonable design goal is for SRAM to be accessible in 2 clock cycles. This means that any memory access can be handled in two pipeline stages. A multi-banked SRAM can accommodate a mix of different access types, including vector access and instruction fetches. If 2-cycle access is not feasible, caches will still be needed for scalar operations and instructions, but vector accesses could access the SRAM directly.

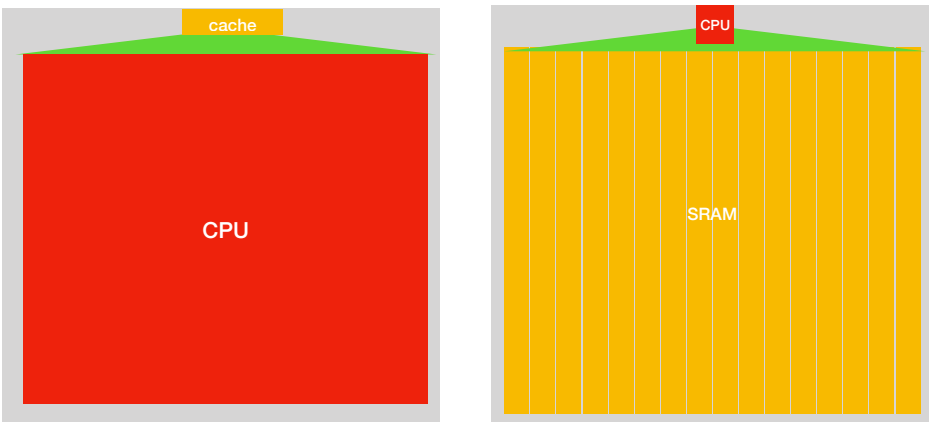
To give some idea of speed, a commodity SRAM from Renases (one of the larger manufacturers of commodity SRAM) is available with a 6 ns clock cycle time and can set up a read or write including the address in one cycle and transfer data in the next cycle. It can also deliver data in burst mode, with up to 4 accesses for one addressing operation [39]. This kind of SRAM is only available in relatively small units, up to a few mebibytes, since the cost per bit is so much higher than DRAM. However, in this proposal, chip space that would otherwise be used for logic will instead be used for SRAM, so provided the result is at least as good as an over-CPU design with the same number of transistors, a big SRAM is not an extra cost.

Exactly how fast an SRAM memory is depends on how it is organized. Relatively large caches on recent Intel (64 MiB) and AMD (512 MiB) server-class CPUs for example have a read latency of about 20ns in the L3 cache [46], but the L3 cache has a more complex access protocol than an ordinary memory, particularly as it is part of a multiprocessor design. What we can be sure about however is that the proposed on-chip SRAM will be faster than an off-chip DRAM.

Using multiple banks that can be addressed and accessed in a pipelined fashion or simultaneously can increase the effective speed, and an on-chip SRAM will not incur off-chip delays. On-chip SRAM could also be potentially accessed with multiple modes of accessing banks, allowing e.g. streaming for vector loads, while also doing scalar loads out of other banks.

---

[4] Characterising a process by nanometers is not strictly accurate as processes vary: transistors per mm2 is proposed as an alternative [9] though that does not capture the fact that some dies may have a larger ratio of interconnect to logic.



(a) Mostly CPU (b) Mostly SRAM

Fig. 3. Comparison of die area used in an over-CPU design like M1 Max vs. Crayon. Layout is conceptual, to illustrate differences in allocation of transistors.

If there are 16 banks as in the Cray-1, each bank in a 1 Gibyte SRAM would be 64 Mibytes. Making the conservative assumption that a 6ns cycle time is the best achievable with an SRAM of this size, overlapping bank accesses would result in an average access time of $\frac{6}{16}$=0.375ns, corresponding to a clock rate of 2.67GHz. From here on, assume memory cycles are 6ns and CPU cycles are 0.375ns. The effect of 16 banks is to allow up to 16 memory operations to occur in parallel, resulting in those 16 memory accesses completing in 2 memory cycles instead of 64. If access to 16 banks is pipelined, the first access will take 2 memory cycles totalling 12ns or 32 CPU cycles but each thereafter can occur on the next CPU cycle. I use 16 here for illustrative purposes; with an on-chip SRAM there is no reason not to have a higher number of banks.

Separately addressed banks have other benefits. Interference between code and data movements can be reduced, and a mix of different types of data movement including vector and scalar loads and stores can be accommodated.

Using either the pipelined or simultaneous access – or both – would not be difficult to design with SRAM and CPU on a single die. Design of the memory controller logic could be tightly integrated with the CPU and memory. While DRAM may include many modes that appear to fit the requirements of diverse forms of access including streaming and even internal bank structure, it is difficult to achieve the theoretically available performance in practice [14].

Putting it all together, I propose a Cray-style RISC plus vector CPU with a sizeable (e.g., 1 Gibyte) SRAM memory on the same chip, called *Crayon* (for Cray on a chip).

It should be possible to implement all the logic required for the CPU in 100-million transistors (noting the aggressive AMD design mentioned in Section 2 had less than that but lacked a vector unit). If that is the case, logic would be about 0.2% of the transistors and SRAM 99.8%.

Fig. 3 Illustrates how die usage differs in an over-CPU design like M1 Max, in which about 2% of the transistors are used for caches, vs. a Crayon design, where the CPU is

less than 1% of die area with most of the rest used to for SRAM. The illustrations are not meant to be floor plans: in a design like M1 Max, CPUs and caches are not grouped as a single entity. They are however drawn approximately to scale to illustrate the difference in design focus.

4.1 Uses

It is useful to consider possible uses of a CPU with a large included SRAM. A proper detailed study of performance is justifiable if there are clear cases where it could be a good idea.

Design variations. One Crayon on its own would have a substantial memory but one that is far smaller than a typical standalone computer. A more powerful system could be built in various ways. If more processing was needed, more than one CPU plus SRAM Crayon chip could be used. If more memory was needed, external DRAM could be added as another level of memory – possibly as a fast paging device [26]. Since any external DRAM would not be accessed as often as with an over-CPU design, off-chip delays would not be as big a bottleneck.

Closely-coupled Crayon parts forming a multiprocessor node could packaged much like a typical stick of DRAM. For example, 16 such units would contain 16 Gibytes of SRAM and would make for a powerful personal computer. Such nodes could be clustered to form a large-scale system – for example, 1024 Crayons could be configured using 64 such nodes, totalling 1 Tibyte of SRAM and 1024 vector CPUs. Such combinations with a fast interconnect could be seen as a non-uniform memory (NUMA) design [22], or use a distributed memory or distributed shared memory [29] architecture.

Some of these ideas would need operating system work, like treating DRAM as a paging device, or implementing distributed shared memory. In the simplest case, each Crayon device could function independently as part of a network of devices or on its own for an application that did not need more memory. Each node would be a Crayon chip, power supply and network interface.

There is considerable work on speeding up network connectivity because of the memory-speed bottleneck in over-CPU designs [35, 17, 24]. Since their problem is harder than Crayon's, there is no need for special innovation in interconnects to implement a Crayon design.

Use cases. One type of use case is for read-once data streams. A single device of this kind with a network interface could process a stream of data that has to be discarded once processed and pass on the results either to another device or to storage.

Square Kilometer Array (SKA) [6] is an example – data produced when it is working at scale will be too large to store and the faster it is processed, the more can be extracted before the data must be discarded. A network of Crayon devices could fill this need. A Cray-style CPU would be well-suited to the sort of computations needed like Fast Fourier Transforms and deconvolution [42]. Astronomy applications use GPUs; speedups over conventional CPUs are typically a small fraction of the available parallelism [7,

8]. This points to a need to balance memory and CPU so it would be worth exploring more such applications as workloads of interest for a Crayon design.

Another example is a malware scanner or a firewall. Data passing through needs to be checked quickly and passed on if no problem is detected. The throughput characteristics of a Crayon device would support such a use. The need for wire-speed firewalls is extending to new applications like automotive [49] – so a relatively low cost device capable of processing complex rules in real time could have a large market. For embedded applications, a variant with a smaller SRAM would fit a lower cost and lower energy requirement.

Many high-performance computing applications were implemented successfully on vector computers, and there is ongoing research into achieving good speedup on such architectures, with good speedups reported for most of the SPECfp2006 benchmark suite [19].

In general, any application that is reasonably partionable and for which vectorizable code can be written could fit a network of Crayon devices. The big advantage over traditional large-scale supercomputers is that the design is scalable. The advantage over commodity CPUs is that a vector design with a high-speed memory is a proven architecture for high-performance computing.

5 Conclusions

The basic principles for designing a Crayon device are relatively straightforward. The Cray-1 and successors contain a range of useful ideas that would be much easier to implement on a single chip. While 1 Gibyte is big compared with RAM on early Cray machines, it is not big by current standards. Nonetheless a Crayon design could be a good starting point for scaling up to large-scale systems. A system with 1024 Cray-class CPUs and 1 Tibyte of SRAM is a substantial computational resource. If this is a good idea, why did the original Cray vector line die out? It had no mass market. If the next big design didn't sell, the company risked bankruptcy. If the proposed Crayon part had sufficient applications, this problem would fall away.

Compared with the original Cray designs, Crayon is a scalable building block and does not require complex manufacturing to build a whole system. A node for a highly partionable computation can be as simple as a Crayon chip, a network interface, and a simple power supply.

Another important detail is power consumption; there is no reason in principle that a Crayon should not be competitive with an over-CPU design. SRAM can wake up fast from low-power modes for example, so the entire memory need not be running on full power at once.

What I present here is a starting point for re-evaluating the trend towards cramming more and more CPU onto each chip. There are many other ways the vast number of transistors available on a die could be used. However, this project focuses on the detail of a Crayon design to evaluate how far it can be taken. Issues to consider include how fast on-chip SRAM can reasonably be accessed, whether caches are still needed or

whether a multi-banked cache obviate that, and details of the instruction set design and how they interact with the memory system.

Getting all the details right of course is nontrivial. The operating system will need to understand the memory hierarchy (that depends how it is designed, e.g., a DRAM layer functioning as a paging device, or distributed shared memory would require more work than a standalone node or a NUMA architecture). Provided that the RISC-V vector architecture is used, compilers will not be a problem. Overall, compared with the original Cray vector machine project, implementing Crayon should be relatively easy.

Future work includes a more detailed design, performance studies and more details analysis of potential use cases.

Acknowledgements

I thank Justin Jonas for sharing details of the SKA project.

References

1. Adit, N., Sampson, A.: Performance left on the table: An evaluation of compiler autovectorization for RISC-V. IEEE Micro **42**(5), 41–48 (2022). https://doi.org/10.1109/MM.2022.3184867
2. Asanović, K., Patterson, D.A.: Instruction sets should be free: The case for RISC-V. Tech. Rep. UCB/EECS-2014-146, EECS Department, University of California, Berkeley (2014)
3. August, M.C., Brost, G.M., Hsiung, C.C., Schiffleger, A.J.: Cray X-MP: The birth of a supercomputer. Computer **22**(1), 45–52 (1989). https://doi.org/10.1109/2.19822
4. Bell, G.: Bell's law for the birth and death of computer classes. Communications of the ACM **51**(1), 86–94 (2008). https://doi.org/10.1145/1327452.1327453
5. Bhati, I., Chang, M.T., Chishti, Z., Lu, S.L., Jacob, B.: DRAM refresh mechanisms, penalties, and trade-offs. IEEE Transactions on Computers **65**(1), 108–121 (2015). https://doi.org/10.1109/TC.2015.2417540
6. Chrysostomou, A., Taljaard, C., Bolton, R., Ball, L., Breen, S., van Zyl, A.: Operating the Square Kilometre Array: the world's most data intensive telescope. In: Observatory Operations: Strategies, Processes, and Systems VIII. vol. 11449, pp. 156–170. SPIE (2020). https://doi.org/10.1117/12.2562120
7. Cotton, W.: GPU-based visibility gridding for faceting. Tech. Rep. OBIT DEVELOPMENT MEMO SERIES NO. 73, National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA (2022), https://www.cv.nrao.edu/ bcotton/ObitDoc/GPUGridv2.pdf
8. Cotton, W.: MultiGPU-based visibility gridding for faceting. Tech. Rep. OBIT DEVELOPMENT MEMO SERIES NO. 77, National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA (2023), hthttps://www.cv.nrao.edu/ bcotton/ObitDoc/MultiGPUGrid.pdf
9. Courtland, R.: Intel now packs 100 million transistors in each square millimeter. IEEE Spectrum **30** (2017), https://spectrum.ieee.org/intel-now-packs-100-million-transistors-in-each-square-millimeter
10. Cybenko, G., Kipp, L., Pointer, L., Kuck, D.: Supercomputer performance evaluation and the Perfect benchmarks. In: Proceedings of the 4th International Conference on Supercomputing. pp. 254–266 (1990). https://doi.org/10.1145/77726.255163

11. Daleiden, P., Stefik, A., Uesbeck, P.M.: GPU programming productivity in different abstraction paradigms: a randomized controlled trial comparing CUDA and Thrust. ACM Transactions on Computing Education (TOCE) **20**(4), 1–27 (2020). https://doi.org/10.1145/3418301
12. Dandamudi, S.P.: RISC Principles, pp. 39–44. Springer, New York, NY (2005). https://doi.org/10.1007/0-387-27446-4\s\do5(3)
13. Elster, A.C., Haugdahl, T.A.: NVIDIA Hopper GPU and Grace CPU highlights. Computing in Science & Engineering **24**(2), 95–100 (2022). https://doi.org/10.1109/MCSE.2022.3163817
14. Eyerman, S., Heirman, W., Hur, I.: DRAM bandwidth and latency stacks: Visualizing DRAM bottlenecks. In: 2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 322–331. IEEE (2022). https://doi.org/10.1109/ISPASS55109.2022.00045
15. Ezratty, O.: Is there a Moore's law for quantum computing? (2023). https://doi.org/10.48550/arXiv.2303.15547
16. Fetherston, R.S., Shaik, I.P., Ma, S.C.: Testability features of AMD-K6™ microprocessor. In: Proceedings International Test Conference 1997. pp. 406–413 (1997). https://doi.org/10.1109/TEST.1997.639643
17. Fotouhi, P., Werner, S., Lowe-Power, J., Yoo, S.B.: Enabling scalable chiplet-based uniform memory architectures with silicon photonics. In: Proceedings of the International Symposium on Memory Systems. pp. 222–334 (2019). https://doi.org/10.1145/3357526.3357564
18. Gokhale, M., Holmes, B., Iobst, K.: Processing in memory: The Terasys massively parallel PIM array. Computer **28**(4), 23–31 (1995). https://doi.org/10.1109/2.375174
19. Gschwind, M.: Workload acceleration with the IBM POWER vector-scalar architecture. IBM Journal of Research and Development **60**(2-3), 14–1 (2016). https://doi.org/10.1147/JRD.2016.2527418
20. Isen, C., John, L.K., John, E.: A tale of two processors: Revisiting the RISC-CISC debate. In: Computer Performance Evaluation and Benchmarking: SPEC Benchmark Workshop 2009, Austin, TX, USA, January 25, 2009. Proceedings. pp. 57–76. Springer (2009). https://doi.org/10.1007/978-3-540-93799-9\s\do5(4)
21. Johns, M., Kazmierski, T.J.: A minimal RISC-V vector processor for embedded systems. In: 2020 Forum for Specification and Design Languages (FDL). pp. 1–4. IEEE (2020). https://doi.org/10.1109/FDL50818.2020.9232940
22. Liu, Y., Kato, S., Edahiro, M.: Analysis of memory system of tiled many-core processors. IEEE Access **7**, 18964–18977 (2019). https://doi.org/10.1109/ACCESS.2019.2895701
23. Lohr, S.: I.B.M. posts $5.46 billion loss for 4th quarter; 1992's deficit is biggest in U.S. business. New York Times (1993), https://www.nytimes.com/1993/01/20/business/ibm-posts-5.46-billion-loss-for-4th-quarter-1992-s-deficit-biggest-us-business.html
24. Lutz, C., Breß, S., Zeuch, S., Rabl, T., Markl, V.: Pump up the volume: Processing large data on GPUs with fast interconnects. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. pp. 1633–1649 (2020). https://doi.org/10.1145/3318464.3389705
25. Machanick, P.: Project CrayOn: Back to the future for a more general-purpose GPU? In: Proc. 2nd Workshop on Pioneering Processor Paradigms, Vienna, Austria (2018)
26. Machanick, P., Salverda, P., Pompe, L.: Hardware-software trade-offs in a Direct Rambus implementation of the RAMpage memory hierarchy. In: Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 105–114 (1998). https://doi.org/10.1145/291069.291032

27. Mattioli, M.: Meet the family. IEEE Micro **42**(3), 78–84 (2022). https://doi.org/10.1109/MM.2022.3169245
28. Mutlu, O., Ghose, S., Gómez-Luna, J., Ausavarungnirun, R.: A modern primer on processing in memory. In: Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann, pp. 171–243. Springer (2022). https://doi.org/10.1007/978-981-16-7487-7\s\do5(7)
29. Nitzberg, B., Lo, V.: Distributed shared memory: A survey of issues and algorithms. Computer **24**(8), 52–60 (1991). https://doi.org/10.1109/2.84877
30. Olukotun, K., Nayfeh, B.A., Hammond, L., Wilson, K., Chang, K.: The case for a single-chip multiprocessor. In: Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating systems (ASPLOS VII). pp. 2–11 (1996). https://doi.org/10.1145/237090.237140
31. Patsidis, K., Nicopoulos, C., Sirakoulis, G.C., Dimitrakopoulos, G.: RISC-V^2: a scalable RISC-V vector processor. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5. IEEE (2020). https://doi.org/10.1109/ISCAS45731.2020.9181071
32. Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Thomas, R., Yelick, K.: A case for intelligent RAM. IEEE Micro **17**(2), 34–44 (1997). https://doi.org/10.1109/40.592312
33. Patterson, D.A.: Reduced instruction set computers. Communications of the ACM **28**(1), 8–21 (1985). https://doi.org/10.1145/2465.214917
34. Patterson, D.A., Ditzel, D.R.: The case for the reduced instruction set computer. ACM SIGARCH Computer Architecture News **8**(6), 25–33 (1980). https://doi.org/10.1145/641914.641917
35. Pearson, C., Dakkak, A., Hashash, S., Li, C., Chung, I.H., Xiong, J., Hwu, W.M.: Evaluating characteristics of CUDA communication primitives on high-bandwidth interconnects. In: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering. pp. 209–218 (2019). https://doi.org/10.1145/3297663.3310299
36. Perotti, M., Cavalcante, M., Wistoff, N., Andri, R., Cavigelli, L., Benini, L.: A “New Ara” for vector computing: An open source highly efficient RISC-V V 1.0 vector processor design. In: 2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP). pp. 43–51. IEEE (2022). https://doi.org/10.1109/ASAP54787.2022.00017
37. Pool, R.: Off-the-shelf chips conquer the heights of computing. Science **269**(5229), 1359–1361 (1995)
38. Poorhosseini, M., Nebel, W., Grüttner, K.: A compiler comparison in the RISC-V ecosystem. In: 2020 International Conference on Omni-layer Intelligent Systems (COINS). pp. 1–6. IEEE (2020). https://doi.org/10.1109/COINS49042.2020.9191411
39. Renesas Electronics Corporation: IDT71V67602 datasheet, https://www.renesas.com/us/en/document/dst/71v67602-datasheet?r=13449, accessed 13 April 2023
40. RISC-V International: RISC-V “V” vector extension version 1.0 (2021), https://github.com/riscv/riscv-v-spec/releases/download/v1.0/riscv-v-spec-1.0.pdf, accessed 19 April 2023
41. Russell, R.M.: The CRAY-1 computer system. Communications of the ACM **21**(1), 63–72 (1978). https://doi.org/10.1145/359327.359336
42. Scaife, A.: Big telescope, big data: towards exascale with the Square Kilometre Array. Philosophical Transactions of the Royal Society A **378**(2166), 20190060 (2020). https://doi.org/10.1098/rsta.2019.0060

43. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., et al.: Larrabee: a many-core x86 architecture for visual computing. ACM Transactions on Graphics (TOG) **27**(3), 1–15 (2008). https://doi.org/10.1145/1360612.1360617
44. Shalf, J.: The future of computing beyond Moore's law. Philosophical Transactions of the Royal Society A **378**(2166), 20190061 (2020). https://doi.org/10.1098/rsta.2019.0061
45. Singh, V., Singh, S.K., Kapoor, R.: Static noise margin analysis of 6T SRAM. In: 2020 IEEE International Conference for Innovation in Technology (INOCON). pp. 1–4. IEEE (2020). https://doi.org/10.1109/INOCON50539.2020.9298431
46. Velten, M., Schöne, R., Ilsche, T., Hackenberg, D.: Memory performance of AMD EPYC Rome and Intel Cascade Lake SP server processors. In: Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering. pp. 165–175 (2022). https://doi.org/10.1145/3489525.3511689
47. Wright, J.C., Schmidt, C., Keller, B., Dabbelt, D.P., Kwak, J., Iyer, V., Mehta, N., Chiu, P.F., Bailey, S., Asanović, K., Nikolić, B.: A dual-core RISC-V vector processor with on-chip fine-grain power management in 28-nm FD-SOI. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **28**(12), 2721–2725 (2020). https://doi.org/10.1109/TVLSI.2020.3030243
48. Wulf, W.A., McKee, S.A.: Hitting the memory wall: Implications of the obvious. ACM SIGARCH computer architecture news **23**(1), 20–24 (1995). https://doi.org/10.1145/216585.216588
49. Yilmaz, E.: Firewall and Intrusion Detection and Prevention Concept for Automotive Ethernet. Master's thesis, Uppsala University (2020), http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-448449
50. Zangwill, W.I., Kantor, P.B.: The learning curve: a new perspective. International transactions in operational research **7**(6), 595–607 (2000). https://doi.org/10.1111/j.1475-3995.2000.tb00219.x