Microprocessors: The New LUT

Steven A. Guccione
Cmpware, Inc.
Austin, TX (USA)
Steven.Guccione@cmpware.com

Abstract

Hardware design for high performance computing appears to be reaching its limits on several fronts. In the desktop microprocessor world, clock speeds seem to have reached their peak somewhere below 10 GHz. Power consumption has begun to approach, and in some cases surpass, 100 W. These issues, along with the problems of managing designs of approximately a billion transistors have caused one of the most profound changes in microprocessor architecture in decades. Every maker of desktop microprocessors has abandoned the traditional performance scaling approaches which have driven the industry for decades and have opted for so-called multicore designs. Today, it is widely accepted that no future commercial high-performance microprocessors will be built using a single CPU core. It is difficult to overestimate the significance of this change.

A similar trend can has been seen in the ASIC and FPGA worlds. As both power and design complexity issues converge on designers, multiple microprocessor cores are increasingly found in integrated circuit designs. This trend in desktop microprocessors, ASIC and FPGA designs points toward a new reconfigurable architecure based on multiple microprocessor cores working togther in parallel.

Such a system has many advantages over more traditional system design approaches. The design is inherently easier to design, uses a familar programming model, provides high levels of performance with modest power consumption.

In this paper a single chip multiprocessor architecture will be explored in more detail. A programming model will be demonstrated using standard peocessor design tools and implementing a flexible, multiprocessor version of the Advanced Encryption Standard (AES). This implementation will serve as a motivation to explore the limits of exploitable parallelism in such a system, as well as issues of programmability and power consumption.

1. Introduction

Hardware design for high performance computing appears to be reaching its limits on several fronts. In the desktop microprocessor world, clock speeds seem to have reached their peak somewhere below 10 GHz. Power consumption has begun to approach, and in some cases surpass, 100 W. These issues, along with the problems of managing designs of approximately a billion transistors have caused one of the most profound changes in microprocessor architecture in decades. Every maker of desktop microprocessors has abandoned the traditional performance scaling approaches which have driven the industry for decades and have opted for so-called multicore designs [1][2][3]. Today, it is widely accepted that no future commercial high-performance microprocessors will be built using a single CPU core. It is difficult to overestimate the significance of this change.

A similar trend can has been seen in the FPGA world. As both power and design complexity issues converge on FPGA manufacturers and users, an increase in the use of microprocessor cores has been noted. This first occurred in FPGA hardware, as in the case of the multiple PowerPC cores in the Xilinx Virtex-II Pro. Shortly thereafter, processor cores have increasingly appeared in the end-user applications with the popularity of “soft” processor cores such as the Altera NIOS.

Vendors of ASIC processor cores such as Tensilica, Arc and Arm are reporting a similar trend. Customers are now averaging several processor cores per ASIC design, with dozens or even hundreds being reported in high-end designs. A number of small companies have even recently been offering multiple processors in a single device to solve a variety of high performance and / or low power applications.

These of trends all point toward the emergence of a new reconfigurable architecture based on multiple traditional microprocessor cores working in parallel. As with many leading edge hardware technologies, development of supporting software for such architectures has lagged. And perhaps more so than with previous architectures, the form and quality of software and tools for these multicore architectures will be a major factor in the rate of their acceptance.

2. Hardware Design Challenges

The ever increasing number of transistors available on a silicon die presents a continuing challenge to hardware designers. The ability to successfully utilize these new resources to provide better solutions stretches not only the limits of existing architectures and the tools but of techniques used to design modern circuits as well. The most recent wave of problems confronting hardware designers includes some ongoing challenges, including managing larger designs.

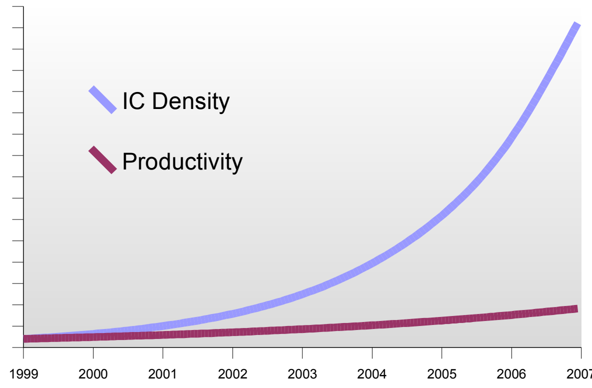


Figure 1: The design gap.

But there are also some new challenges as transistor geometries shrink further into the deep submicron. These include increased power consumption due to leakage currents, signal integrity problems and reliability issues. Finally, the sheer size of modern devices and the high clock speeds present some new limits on designs. A signal may no longer be able to traverse much of the die in a single clock cycle, even with extensive buffering. Such issues are leading to changes in architecture, tools and techniques for hardware design.

2. 1 The Design Gap

The ever-increasing number of transistors available to hardware designers led to a problem commonly referred to as the "Design Gap". This is the implied situation where doubling the available hardware resources should also require a doubling of the amount of labor required to design a large circuit.

With the density of integrated circuits growing at approximately 60% per year and design productivity growing at approximately 20%, there is an increasing gap between the available circuit resources and the human effort required to utilize those circuit resources. Figure 1 illustrates this effect. The exact alignment of the data in the graph is not as important as the fact that managing design complexity becomes increasingly difficult as time passes.

2. 2 Design Abstraction Levels

Fortunately, the management of design complexity has managed to keep pace with the underlying technology through several decades of robust growth. It has not, however, come without substantial changes to the way semiconductor devices are designed. Unlike the underlying silicon technology changes, the changes in design techniques have come less gradually, often by abrupt shifts in design tools and techniques.

Much of the increase in design productivity has come from taking a higher level view of the design. Originally, silicon design was done more or less manually, with design and layout being essentially the same function. Gradually, design was done more abstractly, at the transistor level, with physical design decoupled and often performed by a completely different team.

Era	*Design Unit*	*Transistors*
1960s:	Physical / Transistor	<1
1970s:	Gate	10
1980s:	RTL / Synthesis	1,000
1990s:	IP Blocks	100,000
2000s:	Microprocessors	10,000,000

Table 1: The increasing design abstraction level.

Levels of abstraction rose from transistor to gate to Register Transfer Level (RTL) to larger Intellectual Property or "IP" blocks. Along with this growth, the functionality of the design tools also increased. To further bootstrap this system, the requirements of the design tools and design systems often stretched the limits of performance and functionality of the hardware being designed. Bigger and faster machines were required to build bigger and faster machines.

The increase in circuit density has led to a shift in tools and techniques approximately every decade to decade and a half. This corresponds to roughly a 10x to 100x increase in density, which appears to be the limits on effective use of a given set of design tools and techniques. The process for designing million transistor circuits, for instance, appears to reach its limits at approximately 100 million transistors. The trend has been to raise the level of abstraction, while necessarily retaining the interfaces to the previous levels of abstractions. Table 1 gives a rough indication of the progress of design tools and abstractions over the last half of a century.

This ongoing situation where increasing hardware size and complexity pushed increasingly capable design tools and techniques reached its latest phase at approximately the 10 million transistor level At this point no significant design techniques or tool breakthroughs have occurred in several years since the adoption of IP block design methodologies. Hardware design teams are becoming large enough that the design cost for custom integrated circuits is becoming prohibitive.

2. 3 Verification

While much has been written about the current design gap, a large part of the problem is focused on the verification phase of hardware design. Once the hardware is designed, various techniques are used to test the design to verify that it meets the specification requirements. This typically involves running test vectors on hardware simulators or emulators and comparing them to known good results.

As the size and complexity of hardware has increased, the problem of verifying correct behavior has grown disproportionally. It is estimated that 70% of the overall design effort is currently in verification. But this is somewhat misleading. Even with this large effort, nearly all designs have some defects, and as many as half require a a second pass at design, verification and manufacture to produce an acceptable product. While 70% is the figure often quoted, more resources could clearly be devoted to verification.

2. 4 Power Consumption

Perhaps most importantly, a new emerging design constraint on large circuits is power consumption and heat dissipation. On designs such as desktop microprocessors, the amount of heat generated is in excess of 100W and is reaching the limits of what can be conveniently dissipated. Power budgeting when the design is partitioned is rapidly becoming the dominant design constraint. This concern for power consumption has served to further stretch the capabilities of existing design tools and techniques.

2. 5 The Performance Gap

Finally, the ever increasing size of available hardware and the relative plateauing of microprocessor performance has led to a new and emerging gap in the design space. The raw capabilities of existing hardware now dwarf the raw capabilities for a single traditional processor by many orders of magnitude.

This, combined with the increasing cost of custom hardware, has led to some stark choices in system design. If the design cannot meet its performance goals with software, a large leap in design cost and complexity must be made and a hardware solution implemented. This results in a much higher cost and risk of a custom hardware solution even if the performance requirements only slightly exceed the capabilities of software.

A design requiring twice the performance of a software solution will have to make the leap to the same hardware design environment used by designs that may require many orders of magnitude increase in performance. The large and growing difference in performance between a software and hardware implementation has led to a class of mid-performance solutions that are not well served by existing technology.

3. Multiprocessors

After several years of research [5][6][7][8][9], the movement toward integrating multiple microprocessor cores onto an single semiconductor device has found commercial use relatively suddenly, and in several distinct areas. These areas include desktop microprocessors, Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). It is believed that this represents a larger trend in hardware design overall, indicating the next level of abstraction for hardware design.

First, in the area of desktop microprocessors, after literally decades of successful uniprocessor development, every major vendor of commercial microprocessors has shifted development away from traditional single core processor design to multiprocessors.

For decades, microprocessor architects have been able to take advantage of increasing transistor budgets to increase performance. Various enhancements to the basic microprocessor architecture, including floating point support, multiple ALUs, and on-chip caches have played a large role in increasing performance.

This long standing trend, however, appears to have been broken. The most recent generation of microprocessors has used this latest increase in transistor budget by replicating existing processor architectures on the die.

This represents a very significant break with the past. Up until this point, processors have maintained software compatibility with previous generations. Many processor families offered binary compatibility across decades of products. Many others would at most require a simple re-compilation. Multicore microprocessors, however, have broken this long and impressive trend. At some level, existing software has to be re-written to take advantage of these multicore architectures.

In the FPGA world, the basic cell of modern FPGAs has continued to grow, and now consumes tens of thousands of transistors. Perhaps not coincidentally, this is approximately the same size of a modern embedded processor. Additionally, nearly every FPGA vendor has offered a traditional FPGA fabric with one or more integrated microprocessor cores. This includes the Xilinx Virtex-II Pro [4], the Altera Excalibur and the QuickLogic QuickMIPS. In addition, there has been an strong trend toward using embedded 'soft' processors such as the Xilinx MicroBlaze and the Altera NIOS [15]. It is interesting to note that such 'soft' processors typically take on the order of 1000 LUTs to implement and that on the order of 100 such processors could fit in a modern FPGA device.

Finally, in the ASIC world, reports from microprocessor core vendors is that the average design now uses six microprocessor cores [13] and is increasing. The Cisco Carrier Routing System (CRS-1) reports 188 microprocessors on a single die to do 40 GBPS packet processing [19].

4. Configurable Multiprocessing

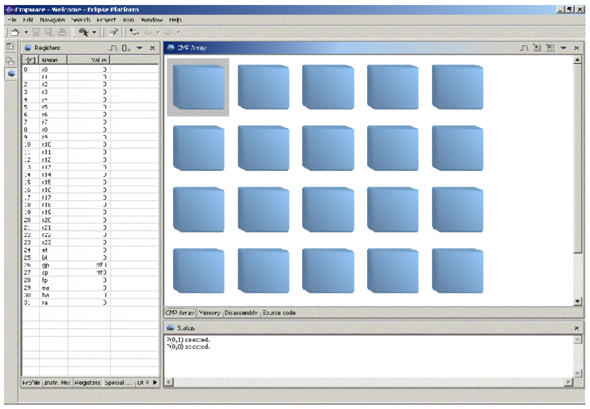
The idea of using a microprocessor as a basic processing element and using many such elements to achieve system design goals is an idea that is gaining popularity. In fact, it is currently possible to configure dozens of processors in an FPGA and to put literally thousands of processors in a single die. This provides a level of raw performance which is orders of magnitude of higher than single core solutions and is competitive with custom hardware. It also provides other capabilities that make it a highly competitive high performance solution. These include:

Flexibility: Multiprocessing on a single device provides a highly programmable solution. The underlying programmability is based on the traditional instruction set model, and thus makes use of traditional compiler technology. This is in comparison to the relatively inflexible hardware design flow that has been the underpinnings of nearly all reconfigurable logic devices.

Power: A multiprocessor is more power efficient than a uniprocessor system. This is one of the reasons commercial desktop microprocessors have moved to multicore approaches. The increasingly complex hardware necessary to boost the performance of a uniprocessor architecture is no longer competitive with multicore approaches.

Simplified hardware design: Perhaps the best reason for the rise of multiprocessing is its ability to manage hardware design complexity. Using existing pre-verified processor cores makes hardware design for such systems relatively easy, if not trivial. This is opposed to going through the risky, expensive and time-consuming custom hardware design and verification process to achieve performance and power goals.

While this approach solves many of the problems facing hardware designers, it does so by moving much of the system design and implementation effort into software. And like so many hardware solutions before it, support for the software challenges has been slower in coming. Because of the pressures due to power, performance and programmability, it appears certain that such multiprocessor systems will be a part of the future of system design. How quickly they are adopted and at what cost will depend largely on the quality and availability of software tools. What these tools will look like is still an open question and one increasingly under study.


Cmpware - Welcome - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Registers
CMP Array
CMP Array | Memory | Disassembly | Source code
Status
Profile | Instr. Mix | Registers | Special ... | Ut

[r]	Name	Value
0	r0	0
1	r1	0
2	r2	0
3	r3	0
4	r4	0
5	r5	0
6	r6	0
7	r7	0
8	r8	0
9	r9	0
10	r10	0
11	r11	0
12	r12	0
13	r13	0
14	r14	0
15	r15	0
16	r16	0
17	r17	0
18	r18	0
19	r19	0
20	r20	0
21	r21	0
22	r22	0
23	r23	0
24	et	0
25	bt	0
26	gp	[illegible]
27	sp	[illegible]
28	fp	0
29	ea	0
30	ba	[illegible]
31	ra	0

```
P(0,1) selected.
P(0,0) selected.
```

Figure 2: The Cmpware CMP-DK.

One set of tools addressing the programming of multiprocessor systems in the Configurable Multiprocessor Development Kit (CMP-DK) from Cmpware, Inc. This integrated software development environment is based on the popular Eclipse environment and provides fast simulation models for microprocessors, integrated into a multiprocessor simulation engine. The toolkit lets processors and their interconnection network be quickly and simply defined. Once these models are in place, code compiled using the standard processor software development tools is loaded onto the processors and executed.

The toolkit provides a wide variety of views of the execution to assist in debug and analysis. These view include familiar debugger views such as source code, disassembly, variables, registers and memory. In addition, cycle counts and an estimation of power consumption as well as statistics and live data for the communication links are provided. This interconnection network information is extremely valuable in analyzing the system and is typically absent in similar systems based on uniprocessor debuggers and tools.

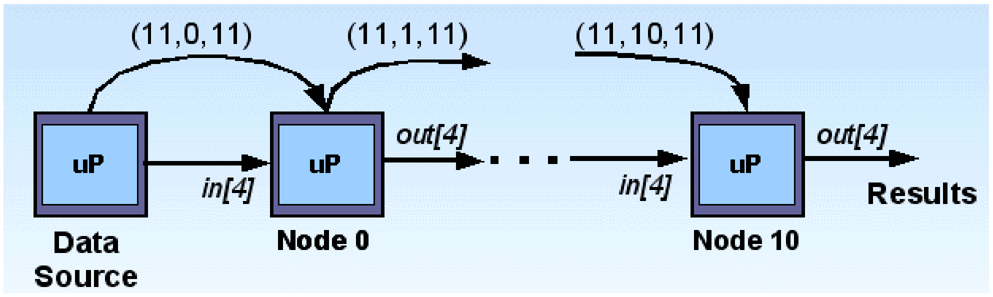


Figure 3: The AES multiprocessor data flow.

The Cmpware toolkit has some default models that are appropriate for many uses. The default processor model is the NIOS II microprocessor core from Altera. The default communication link is a shared 32-bit memory mapped register with a hardware semaphore. This provides a fast and efficient point-to-point communication mechanism for CMP architectures. Data can be written to the register on one cycle and read by another processor on the next. It is this sort of high bandwidth communication that permits processors to exploit parallelism at very fine levels of granularity.

The default network mode is a nearest-neighbor mesh using these shared registers. The dimensions of the processor array, as well as the models for the processor, link and network can be set from an Eclipse preference page and models are constructed and initialized with little or no noticeable delay.

While the Cmpware toolkit can be used to produce any sort of processor / network simulation models, this shared register approach is recommended and has some notable benefits. First, this approach permits the processor / compiler to be treated as a "black box" capable of implementing algorithms from a high level language. It does not require any modification to the processor core, which is likely to be difficult or impossible, depending on the nature of the IP.

Additionally, hardware and architecture modifications will require further modifications to the compiler and other development tools. A memory mapped I/O port requires no hardware modification to the processor core and tools and permits software access via a simple address pointer.

Finally, such communication links are much simpler to analyze and debug than other implementations. Traffic on shared buses, for instance, can be difficult to analyze and problems, particularly race conditions, can be difficult to track down in such environments. With direct communication channels, one processor sends data and another receives it. If some inter-processor communication problem occurs, it is usually a simple matter to find the source.

4. A Programming Model

While there are many proposed programming models for multiprocessors, the availability of high performance communication links between the processors provides a substantial change in the processing characteristics compared to older, system level multiprocessors.

Older system-level multiprocessors had relatively powerful processors and relatively slow communication links. This supported applications with large amounts of processing and short, infrequent inter-processor communications. In general, 'large grained' or task-level parallelism was all that could be exploited.

A configurable multiprocessor has actually reversed this situation. The processor itself can only service a single communication port in a given cycle. But since each processor is likely to have many such ports (the default mesh has four), the configurable multiprocessor actually has more communication bandwidth than processing power.

The programming model currently favored is one similar to hardware development. Functions are designed and implemented in typical serial software fashion, but they are interconnected at a higher level of subroutines that implement the communication. In general, this communication level reads input parameters and data from communication channels and passes this on to the the standard serial subroutines. The results of these subroutines are then sent to other processors as partial or intermediate results for further processing.

While this can be seen as an RTL-like approach, and indeed for the degenerate case of one operation per processor it does resemble RTL very closely, it is possible to perform more general and irregular computation across many cycles in a processor. This provides an implementation platform with much of the power of a custom hardware implementation with very high degree of flexibility.

4. An AES Encryption Example

The Advanced Encryption Standard (AES) is an algorithm for data encryption. This algorithm arose from a proposal by the US National Institute of Standards and Technology (NIST) in 1997 to replace the aging Data Encryption Standard (DES) which was beginning to show signs of vulnerability. AES was selected from a large field of applicants and provides a good balance of simplicity and encryption strength. It also is designed to be implemented efficiently in either hardware or software [16][17].

The algorithm itself takes 128, 192 or 256 bits of data and using a 128, 192 or 254 bit encryption key, produces an encrypted result of the same size as the input. In this example, we will concentrate on the 128 bit input and 128 bit key variation of the algorithm, although all are very similar in implementation.

In this case, the algorithm is broken up into eleven stages called 'rounds'. The first and last round are unique, but the intermediate nine rounds are identical. Each of these rounds takes a 128 bit key and 128 bits of data and produces a 128 bit result. This result is forwarded to the next round of the algorithm.

Such an algorithm can be implemented in hardware with each round pipelined [18]. Similarly, a multiprocessor implementation can implement each round in a single processor, passing temporary results on in a pipeline-like fashion.

Figure 3 shows the multiprocessor implementation of the AES algorithm. The first node in the figure is just used to send test data in to the AES algorithm. This data source is a general input and could be any data source in the system. The arced arrows represent some initialization parameters set to the processors at start-up. These are more explicit in the actual code, but they provide the number of rounds in the algorithm, the number of nodes in the system and the current node number. All nodes begin by reading these parameters, then passing them on to the appropriate neighboring nodes.

The code in Figure 4 is the actual code run on each of the processors. The functionality is relatively simple. Data is read in from the 'west' input port (which is just a pointer and can be given any convenient name). This data is used to call the `round()` function, which is just the standard serial, uniprocessor version of the code used to implement an AES round. Of course, this code is relatively complex and involves various look-up tables, etc., but it can be used unmodified from existing uniprocessor code.

While it is acceptable to provide a point solution for a fixed number of processors, the programmability of a multiprocessor array permits a richer variety of solutions with relatively little effort. Using the parameters passed in to each node, it is possible to configure a solution for a varying number of nodes.

```
for(;;) {

   /* Get input */
   for (i=0; i<4; i++)
      in[i] = *west;

   round(round, in, out);

   /* Send output to next node */
   for (i=0; i<4; i++)
      *east = out[i];

   } /* end for(;;) */
```

Figure 4: The multiprocessor AES code.

For instance, if the parameter giving the number of nodes is '2' and the number of rounds remains necessarily fixed at eleven, it is possible to perform half of the rounds on one node and half on the other. All this will require is another loop around the `round()` function call indicating how many times (and with which parameters) it should be called. This is in fact the way the algorithm is implemented. The addition of the extra looping construct and some brief code to calculate the start and end round for a given processor add approximately a dozen lines of relatively simple code and is not reproduced here.

5. AES Performance Results

Using the fully parameterized code, this AES algorithm was run using the Cmpware toolkit on configuration varying from one to eleven nodes. The default NIOS II processor and shared register mesh network were used. And because the NIOS II development is also Eclipse-based, code development and compilation were done in the same IDE as the multiprocessor simulation. 64k bytes of data were used in each run to get a more realistic estimate of performance. This was done to remove the effect of start-up overheads, in particular key scheduling,

The diagram in Figure 5 shows the speedup as processors are added. The left axis indicates the relative performance in thousands of cycles (KCycles) while the right axis gives the speedup factor. Note that as the number of cycles decreases as processors are added, the speedup increases, as expected. What is striking about this graph, however, is the large gain from adding the second processor. The two-processor implementation runs nearly twice as fast as the single processor implementation. Also of note is the plateau in performance when processors seven through ten are added, then the dramatic speedup when the eleventh is added.

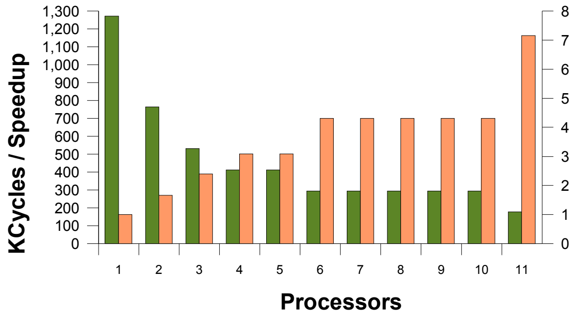


Figure 5: Execution cycles and speedup as processors are added.

This is primarily because of the 'granularity' of the computation. Because the number of rounds is about the same as the number of processors, performance is maximized when there is an even distribution of work. In the cases where some nodes have an extra round to perform, these node become the bottleneck, reducing the overall throughput to that of the slowest processing node.

Figure 6 give a representative execution profile for nine nodes. Clearly this is not the ideal situation for an algorithm with eleven fairly uniform pieces of work to perform. And from the graph it can be seen that nodes five and nine are running at full capacity, while all of the other nodes are running at nearly half. These two nodes have been charged with performing two rounds of computation each, which the remaining node each perform one. A final point is the low utilization of the first node in nearly every case. In the AES algorithm the first round is significantly simpler than the remaining rounds. All that is done is the incoming data is XOR'ed with the encryption key.

Of course, all of these details may be used for further optimizing the system performance. But with eleven processors and eleven rounds the utilization of all of the processor performing the rounds approach 100%. This leads to a factor of seven performance increase over the single processor solution.

What is most notable here is that this speedup occurs on a very programmable platform using very lightly modified existing software. And there is still large amounts of sub-round parallelism exploitable. This would be done by breaking the `round()` method down into component pieces in much the same way that the top-level AES algorithm was partitioned. What is interesting is that this becomes a purely software endeavor, and that large gains in performance are possible with no modification of hardware.

6. AES and Power Consumption

In this example, the processing node selected is a NIOS II processor from Altera. The compiler used is the standard Gnu C shipped with the NIOS development kit. The communication network is a 2D grid, with each processor communicating with its four neighbors, although all of these links may not be used in this particular example. The links used by the processors are Shared Registers, which behave like one word synchronous FIFOs. These permit data to be communicated between nodes in a single cycle, while providing the tight synchronization required for high levels of processor utilization.

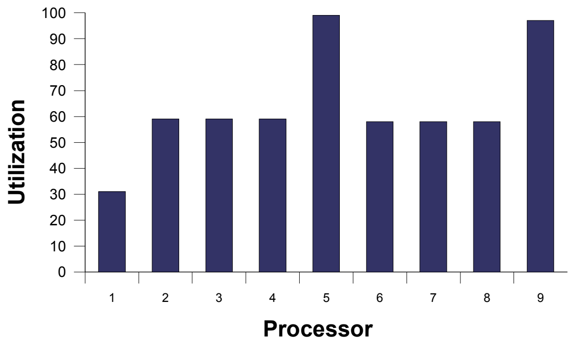


Figure 6: AES utilization profile for nine nodes.

The diagram in Figure 7 illustrates the estimated power consumption as processors are added to the AES encryption algorithm. As with the diagram in Figure 5, the left axis indicates the number of execution cycles in thousands (KCycles) and the right axis gives the estimate power consumption in milliwatts (mW). The actual numbers used in this estimation are the Cmpware defaults of 10 mW while running and 2 mW while idle. This assumes that a processor waiting for communication stalls in an idle mode and consumes a much smaller amount of power than a processor running and performing computation.

The performance data in Figure 7, as indicated by the number of cycles, is identical to that in Figure 5, with the number of total cycles required to execute the algorithm decreasing as processors are added. The profile of the power consumption is very similar to the speedup in Figure 5, and this is to be expected. Because the work is merely being shifted to other processors, the total power consumed should track the processor utilization.

One small effect is seen in the 'plateau' region between processors six and ten. Here processors are added, but the overall performance does not increase. While the utilization stays constant, the power increases slightly. This is due to the power consumption overhead of the idle processors. The added processors, even if they are kept idle and are not contributing to the calculation, still draw some power.

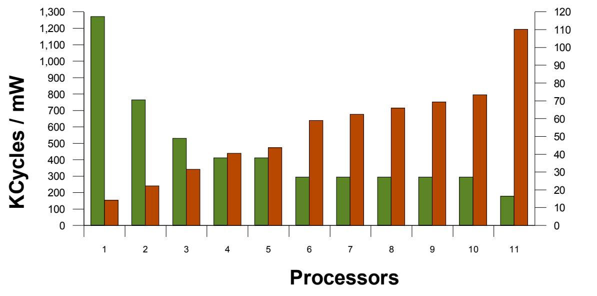


Figure 7: Performance and power consumption as processors are added.

Figure 8 shows a graph of the the total energy consumed, calculated by multiplying the execution time (cycles) by the average power consumption in milliwatts. This graph indicates that there is no particularly large power penalty to adding processors. The inefficiencies in the six to ten processor range, where new processors are added which do not contribute to an increase the speed of the calculation, are seen as the slight rise in this region.

What is most notable is the relatively stable overall energy consumed. Even in this relatively coarse grained calculation, the overall energy consumed varies approximately ten percent from the average. And similar to the data in Figure 6, the addition of processors six though ten did little to increase the overall system performance, while incrementally decreasing the power efficiency.

This trend reverses itself fairly drastically as the workload becomes re-balanced when the eleventh processor is added. The performance not only increases relatively steeply, but the total energy consumed by the calculation actually decreases. This is due to the large increase in the overall efficiency of the processors, reducing the cumulative overhead.

7. CMP and Power Consumption

While the AES implementation is very illustrative of the strengths of the multiprocessor approach to managing performance, some additional features of this approach provide even greater flexibility and even lower power consumption.

First, unlike fixed hardware solutions, this approach can dynamically change the number of processors used, varying the performance and power in a fairly linear fashion. This sort of detailed power management is difficult to achieve in systems with fixed hardware resources.

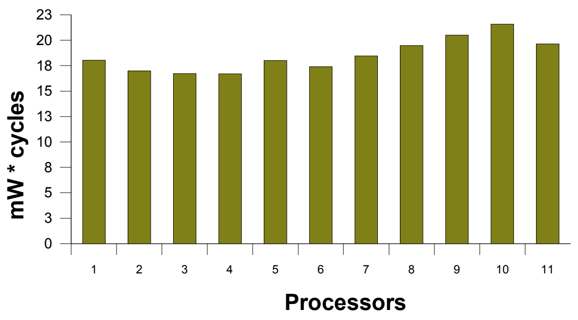


Figure 8: Total energy consumed.

A second feature of this approach is that it is possible to add processors and keep the level of performance constant. This allows a fairly linear reduction in clock speed. In the case of the AES algorithm, eleven processors could perform the algorithm at the same speed as a single processor, but at approximately one-seventh the clock speed.

In the previous graphs, the clock speed was assumed to be a constant. But in most modern microprocessors, the clock speed can be adjusted. In addition, at these lower clock speeds, many modern processor cores can be run at a lower voltage. And because power consumption is proportional to the voltage squared, the power savings can be dramatic, even as hardware is added in the form of more processor cores.

Figure 9 shows the power consumption of the ARM1020E processor from Samsung [20]. Note that as the clock speed is increased, the voltage must also be increased from 0.7V at 400 MHz to 1.1V at 1200 MHz. This factor of three increase in clock speed and performance results in a factor of seven increase in power consumption, given that the voltage must rise with the clock frequency.

A similar effect can also be seen if system performance is held constant and processors are added. As the additional processors raise the level of performance, the clock speed can be lowered. In the general case, this is simply trading the power consumption rate for performance, as described in the AES demonstration. However, if along with this lowering of the clock speed, the voltage is also lowered, additional power savings can be realized.

In the case of the ARM1020E, a single processor running at 1200 MHz has the same level of performance as three processors running at 400 MHz. But the single processor at 1200 MHz will consume 1800 mW, while the three combined 400 MHz processors will only consume 780 mW. This reduces the power consumption by a factor of 2.3, while keeping performance constant.

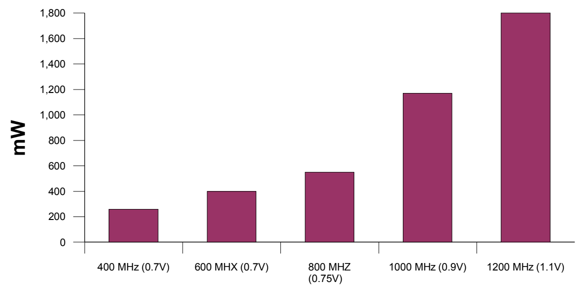


Figure 9: Power and clock speed for the ARM1020E.

Even if the utilization of these processors is less than 100%, reducing the power consumption by more than a factor of two is noteworthy. Particularly because it occurs by more than tripling the amount of hardware used by the system and providing a flexible, fully programmable system.

9. Managing Software

The idea of building hardware with dozens, hundreds or even thousands of processors is not difficult to imagine. In fact, the era of such embedded designs has already begun. While the hardware techniques are clearly within the grasp of today's tools and technologies, application software development will remain the challenge.

The success of application development will depend heavily not only on sufficiently powerful tools, but also on techniques to manage the increased complexity of the software. Many large-grained CMP architectures for desktop processing currently use a parallel task or thread model. Here, the operating system allocates tasks or threads to the various processors. This assumes the system will have a significant amount of task-level parallelism and that the communication between these tasks is either very limited or nonexistent.

The CMP approach of port connected arrays of processors puts further demands on the software, but also has the potential to provide new levels of flexibility and performance. This approach takes more of a hardware view of the world, where data is processed by one unit, then partial results passed on to other units for further processing. This style of programming not only permits extraction of very large amounts of parallelism (as can be seen in hardware implementations), but it also provides a level of flexibility unavailable in fixed hardware solutions.

Such a programmable solution can be re-programmed and re-defined at any time. This allows systems, even fielded systems, to be upgraded to repair bugs, decrease power consumption, increase performance and add functionality. In fact, upgrades may optimize some or all of these parameters simultaneously. This not only extends product lifetimes, but should reduce time to market. In addition to the simplification of the hardware design effort, early systems can be released with a foreknowledge that they will be enhanced via upgrades.

Finally, the implementation of algorithms such as AES has led to the exploration of other issues in the management of software for such large scale parallel systems. One particular problem is the allocation of software routines to the processing nodes.

At the top level, techniques from hardware are again borrowed. A form of 'floor planning' permits blocks of processors to be allocated to particular tasks. For instance, in the case of AES, a block of eleven processors could be allocated for encryption of a channel, and eleven more for the decryption. These 22 processors may or may not all be used to perform the task at hand, but they can be physically reserved.

Similarly, the mechanics of loading particular executable code onto specific nodes can become a chore. With dozens or more nodes, having a separate object for each quickly becomes difficult to manage. As in the AES example, we have favored a technique of 'parameterization' where identical code is loaded into all nodes, and run-time parameters are used to define the detailed functionality.

This is not only similar to function call parameters in software and parameterized macros in hardware, but also has interesting analogs in biology. In an organism, each cell contains all of the same digital code (DNA), no matter what its function. The process of 'differentiation' permits cells to take on more specialized roles, usually based on signals from neighboring cells. Parameterization uses a very similar technique of providing data to neighboring processors to not only manage the software complexity, but to provide a variety of functions for each processor.

10. Conclusions

Today, it is possible to build and program devices containing thousands of processors. Even reconfigurable logic devices such as FPGAs can easily support hundreds of soft processor cores in a single device. This provides the potential for thousands of MIPS of computing power, programmed in traditional high level languages. To make this approach even more attractive, it operates at a power efficiency that is significantly higher than the approach used by existing uniprocessors.

While the hardware is relatively straight forward, the software for such systems will continue to be a challenge. Tools which provide a high level of visibility into the internal workings of such architectures will be necessary. And these tools must be well designed to present the large amount of information generated at run time by such parallel systems.

Lastly, the programming techniques used to exploit these high performance systems will be crucial. Techniques that leverage existing work and have a theoretical basis similar to existing popular languages and styles help make the transition to configurable multiprocessing easier. While the hardware trends are clear in the microprocessor, ASIC and even FPGA areas, how software supporting such systems emerges will define the cost and rate of acceptance of these systems.

11. References

[1] "Intel Demonstrates Breakthrough Processor Design", http://www.intel.com/pressroom/ archive/releases/ 20010828comp.htm, August 28, 2001.

[2] "AMD Announces Technology Milestone With Its Multiple-Core Strategy", http://www.amd.com/us-en/Corporate/VirtualPressRoom/ 0,,51_104_543~86455,00.html, June 14, 2004.

[3] "Sun Drives Multithreaded Processor Innovation with New UltraSPARC IV+", http://www.sun.com/smi/Press/sunflash/2004-10/sunflash.20041005.2.html, October 5, 2004.

[4] "Xilinx enhances FPGAs with embedded PowerPCs", http://www.eet.com/semi/news/ OEG20020304S0017, March 4, 2002.

[5] Lance Hammond, Basem A. Nayfeh and Kunle Olkotun, A Single-Chip Multiprocessor, Computer, Volume 30, Number 9, September 1997, Pages 79-85.

[6] Lance Hammond, Ben Hubbert , Michael Siu, Manohar Prabhu, Mike Chen , and Kunle Olukotun. The Stanford Hydra CMP, IEEE MICRO Magazine, March-April 2000.

[7] "The Future of Microprocessors", David Patterson, U. California at Berkeley, June 2001. http://www.cs.berkeley.edu/~pattrsn/talks/NAE.ppt

[8] Chip Multiprocessing Resources: http://www.princeton.edu/~jdonald/research/cmp/

[9] Stanford Hydra Project: http://www-hydra.stanford.edu/

[10] John Goodacre, Understanding the Options for Embedded Multiprocessing, ARM IQ Journal, Vol.2, No.2.

[11] "Tensilica clears path to multiprocessor SoCs", http://www.eetimes.com/story/OEG20020826S0024, August 26, 2002.

[12] "Tensilica Introduces Industry's First Integrated Development Environment for Multiple Processor SOC Hardware and Software Design", http://www.tensilica.com/html/pr_2003_06_16a.html , June 16, 2003.

[13] "ARC International's Customers Lead Way in Multiprocessing Design", http://www.us.design-reuse.com/news/news5747.html, June 18, 2003.

[14] "Eclipse Platform Technical Overview", http://www.eclipse.org/whitepapers/eclipse-overview.pdf, February, 2003.

[15] "Literature: NIOS II Processor", http://www.altera.com/literature/lit-nio2.jsp, 2004.

[16] J. Daemen, V. Rijmen, ``The Block Cipher Rijndael," *Smart Card Research and Applications, LNCS 1820,* J.-J. Quisquater and B. Schneier, Eds., Springer-Verlag, 2000, pp. 288-296.

[17] J. Daemen and V. Rijmen, ``Rijndael, the advanced encryption standard," *Dr. Dobb's Journal* , Vol. 26, No. 3, March 2001, pp. 137-139.

[18] Scott McMillan and Cameron Patterson, "JBits Implementation of the Advanced Encryption Standards (Rijndael)", *Field-Programmable Logic and Applications*, pages 162-171. Springer-Verlag, Berlin, August 2001. Proceedings of the 11th International Workshop on Field-Programmable Logic and Applications, FPL 2001. Lecture Notes in Computer Science 2147.

[19] "Cisco and IBM collaborate to design and build world's most sophisticated, high-performance 40Gbps custom chip", http://www-03.ibm.com/ chips/news/2004/0609_cisco.html, June 9, 2004.

[20] "Samsung Twists ARM Past 1 GHz", Information Quarterly, Volume 1, Number 1, 2002. (Reprinted from "The Microprocessor Report").

[21] Ian Page, "Compiling Software to Gates", Embedded.com, http://embedded.com/ showArticle.jhtml?articleID=55801142, December 2004.