Chapter 2
Many-Core Architecture for NTC: Energy Efficiency from the Ground Up

Josep Torrellas

Abstract The high energy efficiency of NTC enables multicore architectures with unprecedented levels of integration, such as multicores that include 1000 sizable cores and substantial memory on the die. However, to construct such a chip, we need to fundamentally rethink the whole compute stack from the ground up for energy efficiency. First of all, we need techniques that minimize and tolerate process variation. It is also important to conceive highly-efficient voltage regulation, so that each region of the chip can operate at the most efficient voltage and frequency point. At the architecture level, we want simple cores organized in a hierarchy of clusters. Moreover, techniques to reduce the leakage power of on-chip memories are also needed, as well as dynamic voltage guard-band reduction in variation-afflicted on-chip networks. It is also crucial to develop techniques to minimize data movement, which is a major source of energy waste. Among the techniques proposed are automatically managing the data in the cache hierarchy, processing in near-memory compute engines, and efficient fine-grained synchronization. Finally, we need core-assignment algorithms that are both effective and simple to implement. In this chapter, we describe these issues.

Introduction

As semiconductor devices continue to shrink, it is clear that we are about to witness stunning levels of integration on a chip. Sometime early in the next decade, as we reach 7 nm, we will be able to integrate, for example, 1000 sizable cores and substantial memory on a single die. There are many unknowns as to what kind of architecture such a many-core chip should have to make it general purpose. What is clear, however, is that the main challenge will be to make it highly energy efficient. Energy and power consumption have emerged as the true limiters to developing more capable architectures.

J. Torrellas (✉)
University of Illinois, Urbana-Champaign, Champaign, IL, USA
e-mail: torrellas@cs.uiuc.edu

© Springer International Publishing Switzerland 2016

M. Hübner, C. Silvano (eds.), *Near Threshold Computing*,
DOI 10.1007/978-3-319-23389-5_2

Supply voltage (V_{dd}) reduction is the best lever available to increase the energy efficiency of computing. This is because V_{dd} reduction induces a quadratic reduction in dynamic energy, and a larger-than-linear reduction in static energy. As we have seen in Chap. 1, there is experimental evidence that the most energy-efficient operating point corresponds to a V_{dd} value slightly above the threshold voltage (V_{th}) of the transistor, in what is called the NTC regime [1–4]. Hence, we expect that NTC operation will be most appropriate for these many-cores, as they are designed for energy efficiency from the ground up.

There are some aspects of these many-core architectures that are clear. One is that they will need to have efficient support for concurrency, as transistor integration in the chip will enable massive parallelism. In addition, they will try to minimize data transfers—since moving data around is a major source of energy consumption. Finally, they will have to rely on new technologies that will come online in the next few years. These technologies include efficient on-chip V_{dd} regulation, 3D die stacking, resistive memories, and photonic interconnects, to name a few.

Perhaps less obvious is that all the layers of the computing stack will have to be designed for energy efficiency, and that new energy-efficiency techniques that cut across multiple layers will have to be designed. In this chapter, we outline some of the challenges that these many-cores will have to face, and some of the techniques that can be used to address them. Specifically, after a brief background section, we consider the chip substrate at the level of devices and circuits, the architecture layer, data movement issues, core assignment, and the programming layer.

Background

For several decades, the processor industry has seen a steady growth in CPU performance, driven by Moore's Law [5] and Classical (or Dennard) scaling [6]. Under classical scaling, the power density remains constant across semiconductor generations. Specifically, consider the dynamic power (P_{dyn}) consumed by a certain number of transistors that fit in a chip area A. The dynamic power is proportional to $C \times V_{dd}^2 \times f$, where C is the capacitance of the devices and f is the frequency of operation. Hence, the power density is proportional to $C \times V_{dd}^2 \times f / A$. As one moves to the next generation, the linear dimension of a device gets multiplied by a factor close to 0.7. The same is the case for V_{dd} and C, while the f gets multiplied by 1/0.7. Moreover, the area of the transistors is now $0.7^2 \times A$. If we compute the new power density, we have $0.7C \times (0.7V_{dd})^2 \times f/(0.7^3 \times A)$. Consequently, the power density remains constant.

Unfortunately, as the feature size decreased below 130 nm over a decade ago, classical scaling ceased to apply for two reasons. First, V_{dd} could not be decreased as fast as before. In fact, in recent years, it has stagnated around 1 V, mostly due to the fact that, as V_{dd} gets smaller and closer to the V_{th} of the transistor, the transistor's switching speed decreases fast. The second reason is that static power became significant. The overall result is that, under real scaling, the power density of a set

of transistors increases rapidly with each generation—making it progressively harder to feed the needed power and extract the resulting heat.

In addition, there are further concerns at both ends of the computing spectrum. At the high end, data centers are affected by large energy bills while, at the low end, handheld devices are limited by the energy that can be stored and supplied by batteries.

Energy-Efficient Chip Substrate

While there are still some unclear aspects with NTC, it can potentially decrease the power consumption by more than 40 times [2, 3]. This is a substantial reduction, and implies that many more cores can now be powered-on in a given power-constrained chip. Unfortunately, there are well-known drawbacks of NTC. They include a lower switching speed (possibly ten times lower), and a large increase in process variation—the result of V_{dd} being close to V_{th}. It is possible that researchers will find ways of delivering NTC devices of acceptable speed. However, at the architecture level, the issue of dealing with high process variation is especially challenging.

The Effects of Process Variation

Process variation is the deviation of the values of device parameters (such as a transistor's V_{th}, channel length, or channel width) from their nominal specification. Such variation causes variation in the switching speed and the static power consumption of nominally-similar devices in a chip. At the architectural level, this effect translates into cores and on-chip memories that are slower and consume more static power than they would otherwise do.

To see why, consider Fig. 2.1. Chart (a) shows a hypothetical distribution of the latencies of dynamic logic paths in a pipeline stage. The X axis shows the latency,

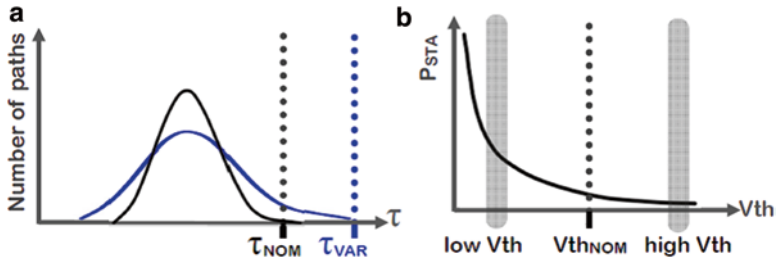

Fig. 2.1 Effect of process variation on the speed (**a**) and static power consumption (**b**) of architecture structures

while the Y axis shows the number of paths with such latency. Without process variation (taller curve), the pipeline stage can be clocked at a frequency $1/T_{NOM}$. With variation (shorter curve), some paths become faster, while others slower. The pipeline stage's frequency is determined by the slower paths, and is now only $1/T_{VAR}$.

Figure 2.1b shows the effect of process variation on the static power (P_{STA}). The X axis of the figure shows the V_{th} of different transistors, and the Y axis the transistors' P_{STA}. The P_{STA} of a transistor is related to its V_{th} exponentially with P_{STA} being proportional to $e^{-V_{th}}$. Due to this exponential relationship, the static power saved by high-V_{th} transistors is less than the extra static power consumed by low V_{th} transistors. Hence, integrating over all of the transistors in the core or memory module, total P_{STA} goes up with variation.

Process variation has a systematic component that exhibits spatial correlation. This means that nearby transistors will typically have similar speed and power consumption properties. Hence, due to variation within a chip, some regions of the chip will be slower than others, and some will be more leaky than others. If we need to set a single V_{dd} and frequency for the whole chip, we need to set them according to the slowest and leakiest neighborhoods of the chip. This conservative design is too wasteful for our energy-efficient NTC many-cores.

Multiple Voltage Domains

NTC chips will be large and heavily affected by process variation. To tolerate process variation within a chip, the most appealing idea is to have multiple V_{dd} and frequency domains. A domain encloses a region with similar values of variation parameters. In this environment, we want to set a domain with slow transistors to higher V_{dd}, to make timing. On the other hand, we want to set a domain with fast, leaky transistors to lower V_{dd}, to save energy. For this reason, extreme scale NTC chips are likely to have multiple, possibly many, V_{dd} and frequency domains. How these domains are selected and set requires considering many trade-offs [7]. We discuss some of these tradeoffs in a later chapter.

However, current designs for V_{dd} domains are energy inefficient [8]. First, on-chip Switching Voltage Regulators (SVR) that provide the V_{dd} for a domain have a high power loss, often in the 10–15 % range. Wasting so much power in an efficiency-first environment is hardly acceptable. In addition, small V_{dd} domains are more susceptible to variations in the load offered to the power grid, due to lacking as many averaging effects as a whole-chip V_{dd} domain. These variations in the load induce V_{dd} droops that need to be protected against with larger V_{dd} guard-bands [9]—also hardly acceptable in an efficiency-first environment. Finally, conventional SVRs take a lot of area and, therefore, including several of them on chip is unappealing. If, as a result, only a few are included in a large NTC chip, the variation inside the V_{dd} domain itself may negate some of the benefits of setting up the domain in the first place.

What Is Needed

To address these limitations, several techniques are needed. First, the many-core chip needs to be designed with devices whose parameters are optimized for low-V_{dd} operation [10]. For example, simply utilizing conventional device designs can result in slow devices.

Importantly, voltage regulators need to be designed for high energy efficiency and modest area. One possible approach is to organize them in a hierarchical manner [11]. The first level of the hierarchy is composed of one or a handful of SVRs, potentially placed on a stacked die with devices optimized for the SVR inductances, or on the package. The second level is composed of many on-chip low-drop-out (LDO) voltage regulators. Each LDO is connected to one of the first-level SVRs and provides the V_{dd} for a core or a small number of cores. LDOs have high energy efficiency if the ratio of their output voltage (V_O) to their input voltage (V_I) is close to 1. Thanks to systematic process variation, the LDOs in a region of the chip need to provide a similar V_O to the different cores of the region. Since these LDOs take their V_I from the same first-level SVR and their V_O is similar, their efficiency can be close 95 %. In addition, their area is negligible: their hardware reuses the hardware of a power-gating circuit. Such circuit is likely to be already present in the chip to power-gate the core. Finally, level converters between the resulting V_{dd} domains can be designed efficiently, by combining them with latches [12].

To minimize energy waste, the chip should have extensive power gating support. This is important at NTC because leakage accounts for the larger fraction of energy consumption. Ideally, power gating should be done at fine granularities, such as groups of cache lines, or groups of functional units. Fine granularities lead to high potential savings, but complicate circuit design. New algorithms need to be designed, possibly at the runtime system level, to control power gating from the software.

Finally, the architectural-level variation parameters of the chip should be made visible to the runtime or operating system. This includes, for each of the core clusters, the V_{dd} and frequencies supported, as well as the static power consumed. The software can then use this information for better assignment of clusters or cores to jobs.

A Streamlined Architecture

Simple Organization

For highest energy efficiency, the NTC many-core architecture should be mostly composed of many simple, throughput-oriented cores, and rely on highly-parallel execution. NTC substantially reduces the power consumption, which can then be leveraged by increasing the number of cores that execute in parallel—as long as the application can exploit the parallelism. Such cores should avoid speculation and complex hardware structures as much as possible.

Cores should be organized in clusters. Such organization is energy-efficient because process variation has spatial correlation and, therefore, nearby cores and memories have similar variation parameter values—which can be exploited by the scheduler.

To further improve energy efficiency, a cluster typically contains a heterogeneous group of compute engines. For example, it can contain one wide superscalar core (also called latency core) to run sequential or critical sections fast. The power delivery system should be configured such that this core can run at high V_{dd} in a turbo-boosted manner. Moreover, some of the cores may have special functionality, or additional instructions.

Minimizing Energy in On-Chip Memories

A large NTC chip can easily contain hundreds of Mbytes of on-chip memory. To improve memory reliability and energy efficiency, it is likely that SRAM cells will be redesigned for NTC [13]. In addition, such memory will likely operate at higher V_{dd} than the logic. However, even accounting for this fact, the on-chip memories may incur substantial energy losses due to leakage. To reduce this waste, the chip may support power gating of sections of the memory hierarchy—e.g., individual on-chip memory modules, or individual ways of a memory module, or groups of lines in a memory module. In principle, this approach is appealing because a large fraction of such a large memory is likely to contain unneeded data at any given time. Unfortunately, this approach is too coarse-grained to make a significant impact on the total power consumed: to power gate a memory module, we need to be sure that none of the data in the module will be used soon. This situation may be rare in the general case. Instead, what we need is a fine-grained approach where we power-on only the individual on-chip memory lines that contain data that will be accessed very soon.

To come close to this ideal scenario, we can use eDRAM rather than SRAM for the last levels of the cache hierarchy—either on- or off-chip. EDRAM has the advantage that it consumes much less leakage power than SRAMs. This saves substantial energy. However, eDRAM needs to be refreshed. Fortunately, refresh is done at the fine-grained level of a cache line, and we can design intelligent refresh schemes [14, 15].

One approach to intelligent refresh is to try to identify the lines that contain data that is likely to be used in the near future by the processors, and only refresh such lines in the eDRAM cache. The other lines are not refreshed and marked as invalid—after being written back to the next level of the hierarchy if they were dirty. To identify such lines we can dynamically use the history of line accesses [14] or programmer hints.

Another approach to intelligent refresh is to refresh different parts of the eDRAM modules at different frequencies, exploiting the different retention times of different cells. This approach relies on profiling the retention times of different on-chip

eDRAM modules or regions. For example, one can exploit the spatial correlation of the retention times of the eDRAM cells [15]. With this technique and similar ones, we may refresh most of the eDRAM with long refresh periods, and only a few small sections with the conventional, short refresh periods.

Minimizing Energy in the On-Chip Network

The on-chip interconnection network in a large chip is another significant source of energy consumption. Given the importance of communication and the relative abundance of chip area, a good strategy is to have wide links and routers, and power-gate the parts of the hardware that are not in use at a given time. Hence, good techniques to monitor and predict network utilization are important. One characteristic of on-chip networks is that they are especially vulnerable to process variation. This is because the network connects distant parts of the chip. As a result, it has to work in the areas of the chip that have the slowest transistors, and in those areas with the leakeast transistors.

To address this problem, we can divide the network into multiple V_{dd} domains—each one including a few routers. Due to the systematic component of process variation, the routers in the same domain are likely to have similar values of process variation parameters. Then, a controller can gradually reduce the V_{dd} of each domain dynamically, while monitoring for timing errors in the messages being transmitted. Such errors are being detected and handled with already-existing mechanisms in the network. When the controller observes an error rate in a domain that is higher than a certain threshold, the controller increases the V_{dd} of that domain slightly. In addition, the controller periodically decreases the V_{dd} of all the domains slightly, to account for changes in workloads and temperatures. Overall, with this approach, the V_{dd} of each domain converges to the lowest value that is still safe (without changing the frequency).

Figure 2.2 shows an example of the way the V_{dd} of a domain converges to a low, safe V_{dd}. In the figure, time is measured in 50-us epochs, and pointers represent errors.

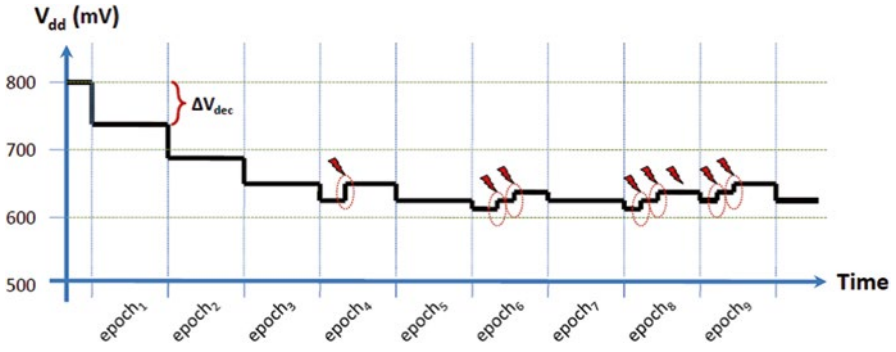

Fig. 2.2 Changes to the V_{dd} of a domain over time

We can see that the reduction in V_{dd} at each step (V_{dec}) gets progressively smaller as time goes by. Moreover, when errors are detected, V_{dd} is increased. With this support, each domain converges to a different low $V_{dd,}$ saving substantial energy in the process. We call this scheme Tangle [16].

Reducing Data Movement

As technology scales, data movement contributes with an increasingly larger fraction of the energy consumption in the chip [17]. Consequently, we need to devise approaches to minimize the amount of data transferred. In this section, we discuss a few mutually-compatible ways to do it.

One approach is to organize the chip in a hierarchy of clusters of cores with memories. Then, the system software can map a program's threads to a cluster and allocate their data in the memories of the cluster. This hardware organization and co-location for locality reduces the amount of data movement needed.

Another technique consists of using a single address space in the chip and directly manage in software the movement of data used by the application in the cache hierarchy. Many of the applications that will run on a 1000-core chip are likely to have relatively simple control and data structures—e.g., performing many of their computation in regular loops with analyzable array accesses. As a result, it is conceivable that a smart compiler performing extensive program analysis [18], possibly with help from the programmer, will be able to manage (and minimize) the movement of data in the on-chip memory hierarchy.

In this case, the architecture supports simple instructions to manage the caches, rather than providing hardware cache coherence transparent to the programmer. Writes do not invalidate other cached copies of the data, and reads return the closest valid copy of the data. While the machine is now certainly harder to program, it may eliminate some data movement inefficiencies associated with the hardware cache coherence—such as false sharing, or moving whole lines when only a fraction of the data in the line is used. In addition, by providing a single address space, we eliminate the need to copy data on communication, unlike what happens in message-passing models.

A third way of reducing the amount of data transfers is to use Processing in Memory (PIM) [19]. The idea is to add simple processing engines close to or imbedded into the main memory of the machine, and use them to perform some operations on the nearby data in memory—hence avoiding the round trip from the main processor to the memory.

While PIM has been studied for at least 20 years, we may now see it become a reality. Specifically, companies are building 3-D integrated circuits that stack one or more dies of memory with a die of logic. For example, Micron's Hybrid Memory Cube (HMC) [20] is a memory chip that contains a die of logic sitting below a stack of 4 or 8 DRAM dies, connected using through-silicon-vias (TSVs). It is easy to imagine how to augment the capabilities of the logic die to support

Intelligent Memory Operations [21]. These can consist of preprocessing the data as it is read from the DRAM stack into the processor chip. They can also involve performing in-place operations on the DRAM data.

Finally, another means of reducing data transfers is to support in hardware efficient communication and synchronization primitives, such as those that avoid spinning in the network. These may include dynamic hierarchical hardware barriers, or efficient point-to-point synchronization between two cores using hardware full-empty bits [22].

The Challenge of Core Assignment

Rationale: Simplicity and Effectiveness

Attaining energy-efficient performance in a many-core with 1000 cores as enabled by NTC requires good core-assignment algorithms. Unfortunately, the number of degrees of freedom in the assignment is vast. Hence, the challenge is to provide effective assignment while keeping the algorithms simple so they are implementable in real systems. In this section, we present a simple algorithm to assign cores to applications.

To keep the algorithm simple, we keep the same V_{dd} for all cores, and assign a different frequency to each application. We set the cluster to be the smallest frequency domain. Clocking all the cores in a cluster at the same frequency is reasonable, since the whole cluster is likely to have a similar value of the systematic component of process variations. To further simplify core assignment, we assign all the cores in the cluster as a group to an application. Any resulting unused cores in the cluster are power gated. Leaving them unused is typically not a problem because, in our environment, there is likely to be a surplus of cores. However, if free cores are scarce, a cluster can take-in multiple applications. Finally, a single application may grab multiple clusters. Such set of clusters is called an Ensemble.

We determine a single V_{ddNOM} for all the cores, and a set of per-cluster f_{max_i} as follows. Each cluster's minimum sustainable voltage, V_{ddMIN}, is set after performing SRAM hold and write stability failure analyses to ensure reliable operation. Then, the chip-wide V_{ddNOM} becomes the maximum of all of the clusters' V_{ddMIN}. After V_{ddNOM} is set, timing tests in the SRAM and logic for each cluster *i* determine the maximum frequency f_{max_i} that the cluster can support at V_{ddNOM}. Such frequency will be the default frequency of the cluster. It can be increased if V_{dd} increases over V_{ddNOM}.

An ensemble runs at a single frequency which, at V_{ddNOM}, is equal to the lowest of the f_{max} of the constituting clusters. We pick a single frequency for the whole ensemble to keep the assignment algorithm simple and, therefore, implementable. In addition, running at a single frequency ensures that all the threads of the application make similar progress, which typically results in faster overall execution. When multiple applications are running concurrently, each is assigned to a different ensemble, which forms a separate frequency domain.

One degree of freedom in the assignment algorithm is whether the clusters that form an ensemble have to be physically contiguous. Not worrying about contiguity simplifies the algorithm, but may result in ensembles where inter-thread communication is expensive. We consider two algorithms, which give either high or low priority to choosing contiguous clusters for an ensemble.

Core Assignment Algorithm: M_Assign

We call this core assignment algorithm *M_Assign* (for many-core assignment) [8]. When a new application arrives, M_Assign assigns an ensemble of clusters to it at a single frequency and, typically, does not revisit the assignment during the application's lifetime. In the following discussion, M_Assign tries to maximize MIPS/w; other related metrics can also be used.

M_Assign uses information from both hardware and application. The hardware information includes each cluster's static power (P_{sta}) and maximum frequency supported (f_{max}) at V_{ddNOM} and reference temperature (T). This information is generated at manufacturing-testing time. Providing this information for a single T may be enough, as at NTC, T is lower than at conventional voltages and does not vary much. However, for higher precision, the manufacturer may provide M_Assign with a table of P_{sta} and f_{max} values for different T. Then, based on on-line measurement of the T, M_Assign would use the most appropriate value. Finally, another piece of information is the load of the chip (which clusters are busy).

The application information is the number of cores requested (equal to the number of threads) and an estimate of the average IPC of the application's threads. The IPC is provided for a few frequency values, and is interpolated for the others. It can be obtained from previous runs of the application or from the current run.

The output of M_Assign is the chosen ensemble of clusters for the application, plus the frequency these clusters should run at—equal to the minimum of the f_{max} of the chosen clusters.

To see the simplicity of M_Assign, assume that an application requests n cores. M_Assign must return an ensemble E of size $|E|$ = ceiling(n/ClSize) clusters, where ClSize is the cluster size. Naively, M_Assign could simply check all the possible groups of $|E|$ free clusters, and pick the group that delivers the maximum MIPS/w at V_{ddNOM}. In our design, M_Assign relies on an intelligent exhaustive search, where the search space gets pruned and the runtime complexity reduced significantly. Specifically, M_Assign repeatedly picks one free cluster i (which can cycle at most at f_{max_i}), and combines it with the best selection of $|E-1|$clusters among those that can cycle faster than i, to arrive at the ensemble E which maximizes MIPS/w:

$$\max_E(\text{MIPS}/\text{watt}) = \min_E(\text{watt}/\text{MIPS}) =$$
$$\min_E\left(\left(\text{sum}_E(P_{sta}) + \text{sum}_E(P_{dyn})\right)/\left(\text{IPC}\times|E|\times\text{ClSize}\times f_{max\_i}\right)\right) =$$
$$\min_E\left(\left(\text{sum}_E(P_{sta}) + C\times V_{ddNOM}^2\times|E|\times f_{max\_i}\right)/\left(\text{IPC}\times|E|\times\text{ClSize}\times f_{max\_i}\right)\right)$$

At the time cluster *i* is considered, all variables of this formula are known except $sum_E(P_{sta})$, which is the total P_{sta} of the ensemble *E* to be formed. We know the frequency of *E*, f_{max_i}, as set by the slowest cluster, namely cluster *i*. The operating V_{dd} is fixed chip-wide to V_{ddNOM}. The number of cores requested determines the ensemble size $|E|$. An estimate of IPC(f_{max_i}) is also available. Finally, C, the average cluster capacitance, is proportional to the area, and does not depend on the selection. $sum_E(P_{sta})$, on the other hand, changes with the selection of the clusters to form *E*. Thus, for each cluster *i* considered, the ensemble that maximizes MIPS/w reduces to the ensemble of the clusters that deliver $\min(sum_E(P_{sta}))$.

M_Assign runs very fast if the clusters are ordered offline from lowest- to highest-consuming P_{sta}, and from highest to lowest f_{max_i}. As M_Assign picks one cluster *i* at a time, it only needs to select, among those with higher f_{max}, the $|E|-1$ ones that have the lowest P_{sta}. It then computes the MIPS/w of the ensemble. This process is repeated once for each available cluster *i*, and the ensemble with the highest MIPS/w is picked.

Programming Challenges

Application software is likely to be harder to write for large-scale NTC many-cores than for conventional machines. This is because, to save energy in data transfers, the programmer has to carefully manage locality and minimize communication. Moreover, the use of low V_{dd} in NTC requires more concurrency to attain the same performance.

An important concern is how users will program these highly-concurrent architectures. In practice, there are different types of programmers based on their expertise. Some are expert programmers, in which case they will be able to map applications to the best clusters, set the V_{dd} and frequency of the clusters, and manage the data in the cache hierarchy well. They will obtain good energy efficiency.

However, many programmers will likely be relatively inexperienced. Hence, they need a high-level programming model that is simple to program and allows them to express locality. One such model is Hierarchical Tiled Arrays (HTA) [23], which allows the computation to be expressed in recursive blocks or tiles. Another possible model is Concurrent Collections [24], which expresses the program in a dataflow-like manner. These are high-level models, and the compiler still has to translate them into efficient machine code. For this, the compiler may have to rely on program auto-tuning to find the best code mapping in these complicated machines.

Conclusion

An NTC many-core will attain major improvements in energy efficiency if we rethink the whole computing stack from the ground up for energy efficiency. In this chapter, we have outlined some of the techniques that can be used. Specifically,

we have discussed the need to provide efficient voltage regulation and support simple cores organized in clusters. Memories and networks can be optimized by reducing leakage and minimizing the voltage guard-bands. Data movement can be minimized by managing the data in the cache hierarchy, processing in memory, and utilizing efficient synchronization. Core assignment to applications needs to be carefully crafted. Finally, a major issue that remains in these machines is the challenge of programmability.

References

1. Kaul H, Anders M, Mathew S, Hsu S, Agarwal A, Krishnamurthy R, Borkar S (2008) A 320 mV 56μW 411GOPS/Watt ultra-low voltage motion estimation accelerator in 65 nm CMOS. In: International solid-state circuits conference, February 2008
2. Chang L, Frank DJ, Montoye RK, Koester SJ, Ji BL, Coteus PW, Dennard RH, Haensch W (2010) Practical strategies for power-efficient computing technologies. In: Proceedings of the IEEE, February 2010
3. Dreslinski RG, Wieckowski M, Blaauw D, Sylvester D, Mudge T (2010) Near-threshold computing: reclaiming Moore's law through energy efficient integrated circuits. In: Proceedings of the IEEE, February 2010
4. Markovic D, Wang CC, Alarcon LP, Liu T-T, Rabaey JM (2010) Ultralow-power design in near-threshold region. In: Proceedings of the IEEE, February 2010
5. Moore GE (1965) Cramming more components onto integrated circuits. Electronics 38(8):114–117
6. Dennard RH, Gaensslen FH, Rideout VL, Bassous E, LeBlanc AR (1974) Design of ion-implanted MOSFET's with very small physical dimensions. IEEE J Solid State Circuits 9(5):256–268
7. Silvano C, Palermo G, Xydis S, Stamelakos I (2014) Voltage island management in near threshold manycore architectures to mitigate dark silicon. In: Conference on design, automation and test in Europe, March 2014
8. Karpuzcu UR, Sinkar A, Kim NS, Torrellas J (2013) EnergySmart: toward energy-efficient manycores for near-threshold computing. In: International symposium on high performance computer architecture, February 2013
9. James N, Restle P, Friedrich J, Huott B, McCredie B (2007) Comparison of split versus connected-core supplies in the POWER6 microprocessor. In: International solid-state circuits conference, February 2007
10. Wang H, Kim NS (2013) Improving platform energy-chip area trade-off in near-threshold computing environment. In: International conference on computer aided design, November 2013
11. Ghasemi HR, Sinkar A, Schulte M, Kim NS (2012) Cost-effective power delivery to support per-core voltage domains for power-constrained processors. In: Design automation conference, June 2012
12. Ishihara F, Sheikh F, Nikolic B (2004) Level conversion for dual-supply systems. IEEE Trans Very Large Scale Integr Syst 12(2):185–195
13. Gemmeke T, Sabry MM, Stuijt J, Raghavan P, Catthoor F, Atienza D (2014) Resolving the memory bottleneck for single supply near-threshold computing. In: Conference on design, automation and test in Europe, March 2014
14. Agrawal A, Jain P, Ansari A, Torrellas J (2013) Refrint: intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies. In: International symposium on high performance computer architecture, February 2013

15. Agrawal A, Ansari A, Torrellas J (2014) Mosaic: exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules. In: International symposium on high performance computer architecture, February 2014
16. Ansari A, Mishra A, Xu J, Torrellas J (2014) Tangle: route-oriented dynamic voltage minimization for variation-afflicted, energy-efficient on-chip networks. In: International symposium on high performance computer architecture, February 2014
17. Kogge P et al (2008) ExaScale computing study: technology challenges in achieving exascale systems. In: DARPA-IPTO sponsored study, DARPA. September 2008
18. Feautrier P (1996) Some efficient solutions to the affine scheduling problem. Part I: One-dimensional time. Unpublished manuscript
19. Kogge P (1994) The EXECUBE approach to massively parallel processing. In: International conference on parallel processing, August 1994
20. Micron Technology Inc. (2011) Hybrid memory cube. http://www.micron.com/products/hybrid-memory-cube
21. Fraguela B, Feautrier P, Renau J, Padua D, Torrellas J (2003) Programming the FlexRAM parallel intelligent memory system. In: International symposium on principles and practice of parallel programming, June 2003
22. Smith BJ (1982) Architecture and applications of the HEP multiprocessor computer system. In: Real-time signal processing IV, pp 241–248
23. Bikshandi G, Guo J, Hoeflinger D, Almasi G, Fraguela BB, Garzaran MJ, Padua D, von Praun C (2006) Programming for parallelism and locality with hierarchically tiled arrays. In: International symposium on principles and practice of parallel programming
24. Budimlic Z, Chandramowlishwaran A, Knobe K, Lowney G, Sarkar V, Treggiari L (2009) Multi-core implementations of the concurrent collections programming model. In: Workshop on compilers for parallel computers