Enabling the Heterogeneous Accelerator Model on Ultra-Low Power Microcontroller Platforms

Francesco Conti*, Daniele Palossi†, Andrea Marongiu*†, Davide Rossi*, Luca Benini*†

*University of Bologna, †ETH Zurich
{f.conti, a.marongiu, davide.rossi, luca.benini}@unibo.it,
{daniele.palossi, a.marongiu, luca.benini}@iis.ee.ethz.ch

***Abstract*—The stringent power constraints of complex microcontroller based devices (e.g. smart sensors for the IoT) represent an obstacle to the introduction of sophisticated functionality. Programmable accelerators would be extremely beneficial to provide the flexibility and energy efficiency required by fast-evolving IoT applications; however, the integration complexity and sub-10mW power budgets have been considered insurmountable obstacles so far. In this paper we demonstrate the feasibility of coupling a low power microcontroller unit (MCU) with a heterogeneous programmable accelerator for speeding-up computation-intensive algorithms at an ultra-low power (ULP) sub-10mW budget. Specifically, we develop a heterogeneous architecture coupling a Cortex-M series MCU with PULP, a programmable accelerator for ULP parallel computing. Complex functionality is enabled by the support for offloading parallel computational kernels from the MCU to the accelerator using the OpenMP programming model. We prototype this platform using a STM Nucleo board and a PULP FPGA emulator. We show that our methodology can deliver up to 60x gains in performance and energy efficiency on a diverse set of applications, opening the way for a new class of ULP heterogeneous architectures.**

I. Introduction

The multi-core revolution is now an established reality in almost every computing domain, from system-on-chip (SoC) designs used in most modern smartphones to the extreme parallelism typical of data-center computing. One of the few classes of computers where single-core systems are still predominant is that of low-power microcontrollers, typically employed for energy-constrained and control-oriented systems that are poorly amenable to be architected as parallel designs, as they are either very constrained in terms of power or require a high degree of predictability. This "traditional" view is now being challenged by new classes of applications, typical of the Internet-of-Things (IoT) domain, that couple the need for low power consumption with a demand for higher performance [1][2]. Examples include algorithms based on machine learning techniques (e.g. in embedded machine vision or voice recognition), compressed sensing (e.g. in biomedical applications) and many others.

A common approach to tackle the limited capability of the main microcontroller unit (MCU) in successful extendable embedded systems (e.g., Arduino [3]) is to allow for simple expansion of the base board through daughter boards providing additional sensing and actuation capabilities, such as microphones, cameras and motors. Arduino's *shields* are a representative example of this approach. We propose to adopt a similar methodology to extend the main MCU with a fully-programmable, ultra-low power (ULP) parallel accelerator: in other words, a general-purpose computing device that is designed to deliver high performance/watt for parallel workloads. We can summarize in three points the key characteristics that distinguish this model from the current state of the art in the sub-10mW space [4][5]:

1) *energy efficiency*: to a much larger degree with respect to what happens in larger scale heterogeneous computers, an ULP accelerator needs to be significantly more energy-efficient than its host to be effective;
2) *dynamic offload*: for the accelerator to be programmable (as opposed to simply reconfigurable), it must be possible to dynamically offload different applications from the host; this marks a strong difference to typical fixed functionality ASICs that are often used in a similar domain;
3) *programmability*: development of accelerator code must rely on a programming model that allows efficient exploitation of parallelism, while leveraging a lightweight runtime with reduced execution overhead and memory footprint.

In this work, we present an embodiment of this general model that is based on a STM32 microcontroller and on PULP, a 28nm FD-SOI programmable ULP parallel platform [6]. The MCU and PULP are loosely coupled via a low-power SPI connection that is used both for controlling the accelerator and for data exchange. We prototyped this system using a Nucleo board featuring a *STM32-L476* Cortex-M4 MCU and a Xilinx Zynq-ZC706 board that hosts a functionally complete FPGA emulator of PULP. On top of the SPI protocol that is used to connect the two boards, we implemented a lightweight software abstraction for host (MCU) to accelerator (PULP) communication. A streamlined implementation of the OpenMP runtime library executes on top of the parallel cores in PULP, providing a convenient programming interface to end-users.

The third embodiment of the PULP platform (PULP3) is currently under fabrication in STM FD-SOI 28nm technology, which allowed us to develop a simple yet accurate power model based on post-layout power annotation for the taped-out chip. Our experiments explore power and performance results for the acceleration of a diverse set of computation kernels, including real benchmarks from the computer vision, machine learning, and linear algebra domains, showing that it is indeed possible to use a programmable accelerator to achieve a speedup of more than one order-of-magnitude in the sub-10mW space.

II. Related work

Most commercial microcontrollers target relatively high performance levels at a 50-100 mW power budget such as the STMicroelectronics *STM32F40x* family, based on the ARM Cortex-M4 core [7][8] or the NXP *LPC1800* series [9]. Only a few state-of-art relatively high performance ULP microcontrollers are able to work in a power budget of less than 50 mW: examples include the SiliconLabs *EFM32* [10], the Texas Instruments *MSP430* [11] series of MCUs, the *Ambiq Apollo* [4], and the STMicroelectronics *STM32-L476* [12]. Most of these commercial ULP microcontrollers exploit the fact that CMOS technology is inherently most efficient when driven at near-threshold voltage [13]; the same applies to research microcontrollers. For example Ickes et al. [14], *SleepWalker* [15] and *Bellevue* [16] show examples of near-threshold ultra-low power microcontrollers, with the latter also exploiting SIMD parallelism to improve performance. Singhal et al. [17] propose to use non-volatile memory to achieve a significant reduction of both active and standby power.

None of the platforms mentioned above targets a performance

978-3-9815370-6-2/DATE16/

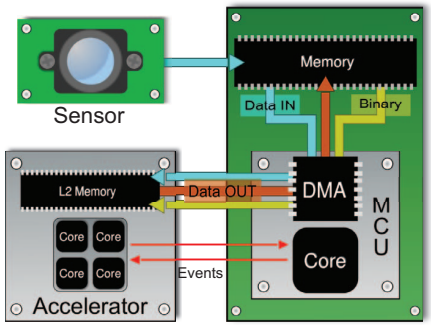


Figure 1: Heterogeneous system composed by a sensor, a MCU and an accelerator.

level higher than a few tens of MOPS in the ULP domain. Those that do so have to rely on both near-threshold and parallel computing. For example *Centip3de* [18] is a large-scale fabric of clusters of Cortex M3 cores, integrated in a 3D matrix. With 64 cores running at 10 MHz, it can reach a peak performance of 0.64 GOPS. Another similar platform is *DietSODA* [19] that features 128 SIMD lanes working at lower frequency (50 MHz) than the rest of the chip, reaching up to 6.4 GOPS. Dogan et al. [20] explore multicore design in subthreshold for biomedical usage, with a power budget as low as 10µW; however in this case performance is several orders of magnitude below that sought after in our work. On the commercial side, to the best of our knowledge there are only two ULP multicore MCUs on the market: the NXP LPC51400 [21] and the Freescale S12 Series [22]. Both of them employ multiple cores with the objective of distributing heterogeneous tasks to each core to save power, not to reach a significant amount of speedup with respect to single-core solutions.

Past work on integration of low-power microcontrollers with accelerators has mainly concentrated on coupling with special-purpose computing devices such as specialized DSPs [23][24], ASICs [5][25], or reconfigurable computing fabrics [26]. Contrarily to the model we propose, none of these platforms can be considered fully programmable in a general-purpose sense, and no one supports a full offload of code from a host.

III. Architecture

A. Heterogeneous Accelerator Model

In its simplest form, an heterogeneous system can be constituted by an *host* and an *accelerator* that are coupled by means of some kind of link. To satisfy the tight power constraints typical of embedded systems such as IoT nodes, this coupling cannot rely on fast and sophisticated sharing mechanisms between the host and the accelerator. Instead, it must be very cheap in terms of power and simple enough to allow integration in an inexpensive system-on-board. In this work, we consider using a simple SPI or Quad SPI (QSPI) link to implement the host-accelerator coupling and allow the offload of code and the exchange of data between the two. SPI is ubiquitous in off-the-shelf MCUs and satisfies the cost and power constraints of an ULP system. Additionally, we consider that the host and the accelerator can exchange a small set of synchronization events (typically implemented with simple GPIOs).

Figure 1 shows the heterogeneous accelerator model we consider in this work. Since the typical purpose of an IoT node is to elaborate data coming from a sensor, we consider a generic sensor interface as the original source of input data. The sensor is managed by the host MCU, which marshals data to/from the accelerator through the low-power coupling link by means of a DMA controller.

Real applications are generally composed by a sequence of kernels (i.e. basic algorithmic elements). A general mechanism of code offload can therefore consist in the offload of an entire collections of kernels (a library) at the same time, or of the strictly required kernel alone. Due to the limited amount of memory available in typical ULP systems such as the accelerator we consider, we chose to restrict our analysis to this second case. The offload implementation itself can be built on top of low-level primitives that handle host-to-accelerator communication through the coupling link; in our case, this consists in primitives to initialize the SPI and DMA peripherals of the MCU and invoke inbound or outbound DMA transfers through the SPI channel. Synchronization events can be managed by means of wait-for-event primitives or interrupts on both the MCU and the accelerator. However, computation offload is typically exposed to programmers at a higher degree of abstraction than these simple primitives. Following what has been done in Marongiu et al. [27], in our platform we support the OpenMP v4.0 specification, where the `#pragma omp target` directive allows to outline a block of code which needs to be compiled for the target accelerator and the `map` clause allows to specify data items from the host program that need to be made visible to the accelerator. In this way, we provide a distinction between program and data offloads and hide the low-level details of the data exchange primitives behind higher level abstractions.

B. PULP Architecture

The accelerator platform we consider in this work is based on *PULP* (Parallel processing Ultra Low Power platform), a scalable, clustered many-core computing platform designed to operate on a large range of operating voltages, achieving in this way a high level of energy efficiency over a wide range of application workloads [6]. In particular, we focus on the third embodiment of the platform, the PULP3 SoC (shown in Figure 2). PULP3 features a single quad-core cluster integrated with 64 kB of L2 SRAM memory and several IO peripherals accessible through a system bus such as two QSPI interfaces (one master and one slave), GPIOs, a bootup ROM and a JTAG interface suitable for testing. The QSPI interfaces can be configured in *single* or *quad* mode depending on the required bandwidth, and they are suitable for interfacing the SoC with a large set of off-chip peripherals (non volatile memories, voltage regulators, cameras, etc.) or with a host microcontroller, such as in our case.

The PULP cluster has 4 OpenRISC-ISA [28] cores with a power-optimized microarchitecture called OR10N [29] and a shared instruction cache (*I$*). The OR10N core features several enhancements with respect to the reference OpenRISC implementation, including a register-register multiply-accumulate instruction, vectorized instructions for `short` and `char` data types, two hardware loops and support for unaligned load/store

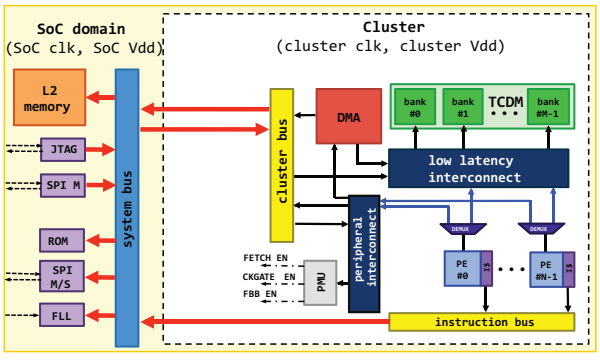


Figure 2: PULP System-on-Chip architecture.

operations. To avoid the energy overhead of memory coherency, the cores do not have private data caches: they all share a L1 multi-banked tightly coupled data memory (*TCDM*) acting as a shared data scratchpad. Intra-cluster communication is based on a high bandwidth *low-latency interconnect*, implementing a word-level interleaving scheme to reduce access contention to the TCDM [30]. A lightweight multi-channel DMA enables fast communication with the L2 memory and external peripherals [31]. The DMA features a direct connection to the TCDM to reduce power consumption by eliminating the need for an internal buffer.

To enable fine grained frequency tuning, a Frequency-Locked Loop [32] and two clock dividers (one for the cluster and one for peripherals) are included in the SoC. Each core can separately be clock-gated to reduce dynamic power or "boosted" by means of a body bias multiplexer. This feature is integrated directly in the thread creation/destruction routine in the runtime to be fully transparent to the user. The cluster also contains a HW synchronizer used to accelerate synchronization between the cores, making sure that they can be put to sleep and woken up in just a few cycles.

C. Prototype

As a prototype of our heterogeneous acceleration approach, we coupled a STM32 Nucleo board with a L467 Cortex-M4 microcontroller and a Xilinx ZC-706 evaluation board featuring a Zynq Z7045 SoC. The Zynq SoC on the Xilinx ZC-706 board hosts the emulator of the PULP chip, which is fully implemented on the Zynq Programmable Logic. The Cortex A9 dual core Processing System available on the Zynq runs Linux and is used exclusively to ease control, introspection and debugging of PULP. The STM32 is connected to PULP via a SPI link; the STM32 acts as the master. All components in the prototyping platform are controllable through a normal workstation, through GDB and UART in the case of the STM32, and passing through the Zynq Processing System in the case of PULP.

The SPI physical link between the Nucleo board and the ZC-706 board has been realized with simple wires connecting the dedicated SPI pins of the Nucleo with a set of pins on the programmable logic of the ZC-706. Although the *STM32-L476* features a QSPI interface, the one on the Nucleo evaluation board does not expose its pins, therefore our physical prototype used one of the single-bit SPI interfaces. Two additional STM32 GPIOs are hooked to the PULP emulator: a fetch enable used to trigger execution of the benchmark; and an end of computation event triggered by PULP and used by the STM32 to resume from sleep.

IV. Results

A. Setup

In this Section we compare execution time, power and energy between the *STM32-L476* microcontroller device, the PULP accelerator and the heterogeneous system composed by the coupling of the two. We base our comparison on a set of benchmarks taken from the fields of linear algebra, learning and machine vision. To provide a fair comparison, all benchmarks are coded in fully portable C, using the OpenMP programming model to express parallelism for PULP, without relying on target specific extensions. To compile the benchmarks, we used the ARM Sourcery Linux GNU toolchain (version 4.8.2) for the ARM Cortex-M target and the OR10N LLVM/Clang toolchain (based on LLVM 3.7) for PULP.

To estimate how much power PULP consumes, we derived our leakage and dynamic power with backannotated switching activities from three power analysis input vectors: *idle* (cores idle), *matmul* (cores running, low pressure on memories) and *dma* (DMA running, high pressure on memories). The FPGA emulation platform is augmented with a performance monitoring unit that is used to measure active and idle cycles for cores, DMAs and interconnects. The average dynamic power consumed over a benchmark is computed from the following model:

$$P_{\mathrm{d}} = f_{\mathrm{clk}} \sum_{i=0}^{N} \left(\chi_{i,\mathrm{idle}} \cdot \rho_{i,\mathrm{idle}} + \chi_{i,\mathrm{run}} \cdot \rho_{i,\mathrm{run}} + \chi_{i,\mathrm{dma}} \cdot \rho_{i,\mathrm{dma}} \right)$$

where χ_i is the ratio of the active cycles of the i-th component over the total benchmark cycles and ρ_i is the dynamic power density. Leakage power, dynamic power density and maximum clock frequency at each operating point are modeled after the post-layout backannotated timing and power analysis results for the PULP3 chip, which was performed at operating points between V_{DD} =0.5 V up to 1 V in 100 mV steps. To estimate maximum frequency at operating points not covered by timing analysis, we used a simple polynomial interpolation model. For what concerns the MCUs, the operating points are those listed in the relevant datasheets [7][8][10][11][4][12][9], using power from the *typical* range. Relying on the similarity between the Cortex-M3 and M4 microarchitectures, we estimated the execution cycles on the Cortex-M3 devices by running the code on the *STM32-L476* with all Cortex-M4 specific flags deactivated. Table I summarizes the benchmarks used in our evaluation. The linear algebra kernels are taken from the standard set of PULP tests; two of them use integers (`char`s and `short`s), while the third uses 16-bit fixed-point numbers. The `svm` kernels are based on a C porting of *libsvm* [33]; they work on 16-bit fixed-point data. The `cnn` ones extend the *CConvNet* library [34] and are also working on 16-bit fixed-point numbers. Finally, `hog` is taken from the *VLFeat* library [35] and it uses 32-bit fixed-point data.

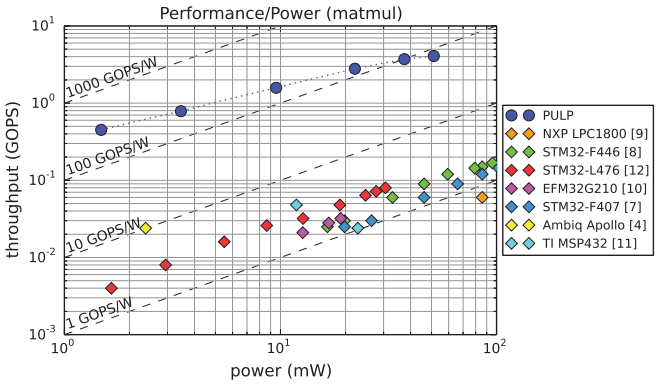


Figure 3: Energy efficiency on the `matmul` test in PULP and several MCUs.

Figure 3 compares throughput in terms of GOPS (billions of RISC operations per second) and power between PULP and several commercial MCUs based on the ARM Cortex-M3 and Cortex-M4 microarchitectures on the `matmul` benchmark[1]. The `matmul` benchmark provides a quasi-ideal case for both parallelization and microarchitectural optimizations; there is a gain of 1.5 orders of magnitude in energy efficiency between PULP and the MCUs that provides an upper bound to the efficiency "slack" available to the accelerator, i.e. to the amount of acceleration that is possible to achieve without wasting power. In absolute terms, the peak energy efficiency shown by PULP

[1]We define the number of RISC operations required by a benchmark by running it on a single OR10N core deactivating all microarchitectural improvements mentioned in Section III-B; in this configuration, essentially equal to that defined in the OpenRISC 1000 ISA [28], OR10N has a very simple 5-stage pipeline and a reduced instruction set, comparable to that of the original MIPS.

Benchmark	Description	Field	Input	Output	Binary Size	RISC ops[1]
`matmul`	Matrix multiplication on `char` data	linear algebra	8 kB	4 kB	11 kB	2.4M
`matmul` (*short*)	Matrix multiplication on `short` data	linear algebra	16 kB	8 kB	11 kB	2.4M
`matmul` (*fixed*)	Matrix multiplication on 16-bit fixed-point data	linear algebra	16 kB	8 kB	13 kB	2.7M
`strassen`	Strassen algorithm for fast matrix multiplication	linear algebra	8 kB	4 kB	6.7 kB	2.3M
`svm` (*linear*)	Support Vector Machine classifier (linear kernel)	learning / vision	6.9 kB	1.6 kB	11.4 kB	650k
`svm` (*poly*)	Support Vector Machine classifier (polynomial kernel)	learning / vision	6.9 kB	1.6 kB	11.5 kB	684k
`svm` (*RBF*)	Support Vector Machine classifier (radial basis function kernel)	learning / vision	6.9 kB	1.6 kB	11.6 kB	781k
`cnn`	Convolutional Neural Network	learning / vision	2 kB	40 B	48.1 kB	3.3M
`cnn` (*approx*)	Convolutional Neural Network (approximated)	learning / vision	2 kB	40 B	48.1 kB	2.6M
`hog`	Histogram of Oriented Gradients feature descriptor	vision	16 kB	36 kB	31.2 kB	31M

Table I: Summary of the benchmark kernels.

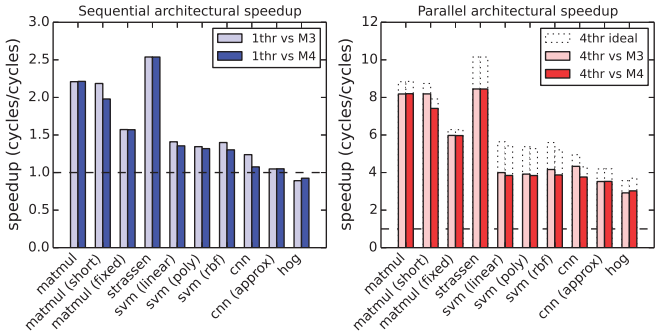


Figure 4: Architectural speedup.

is of 304 GOPS/W with a power consumption of 1.48 mW, whereas that of the MCUs keeps always below 5 GOPS/W at a comparable level of power consumption. The only exception to this is the Apollo Ambiq MCU, which reaches 10 GOPS/W working at a low performance 24 MOPS operating point.

B. Performance evaluation

To better understand the relative contributions to the speedup of microarchitectural optimizations and of parallelization, the left side of Figure 4 shows the *architectural speedup* (defined as the speedup in terms of execution cycles) of our benchmarks running on a single OR10N core in PULP versus the same code on a Cortex M3 or M4. All microarchitectural optimizations available (such as HW loops and pseudo-SIMD vectorization in the case of OR10N) where activated on all targets; moreover, the code was compiled at the maximum compiler optimization level (`-O3`).

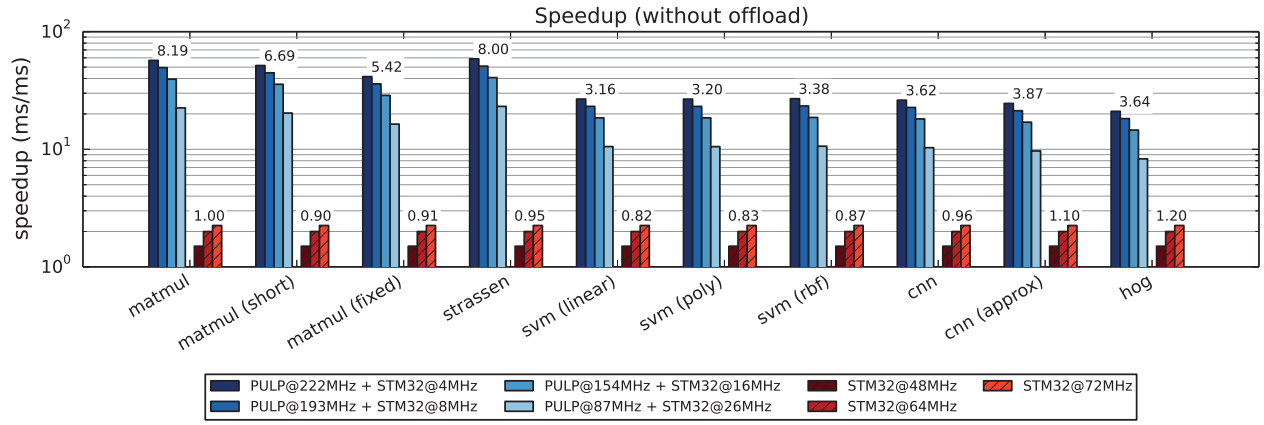
We can divide our tests in three groups: the integer tests (`matmul` on both `char`s and `short`s and `strassen`) show a speedup of 2-2.5x due to the usage of all OR10N optimizations, in particular the register-register MAC instruction, infra-word vectorization and unaligned load/store operations. The tests based on fixed-point computations (all versions of `svm` and `cnn` and the fixed-point version of `matmul`) cannot exploit the OR10N microarchitectural enhancements to the same level, because many of these enhancements are not compatible with fixed-point computation (e.g. there is no multiply-shift-add operation). Finally, the `hog` algorithm has the interesting property of needing a very high dynamic range, and is thus ill-suited to fixed-point implementation; to ensure accuracy is kept at an acceptable level, we had to employ 32-bit fixed-point numbers and SW-emulated 64-bit variables for accumulation. This is the cause of the slight architectural slowdown visible in the figure.

The right side of Figure 4 shows the acceleration gained with code parallelization (on top of the architectural speedup already discussed for the plot on the left); we also show the ideal 4x speedup. This plot essentially shows that OpenMP parallelization is working as it is meant to; the real speedup is reduced by the Amdahl overhead in both the computation code and the runtime; most of this reduction is due to Amdahl non-idealities and the average overhead of the OpenMP runtime is 6%.

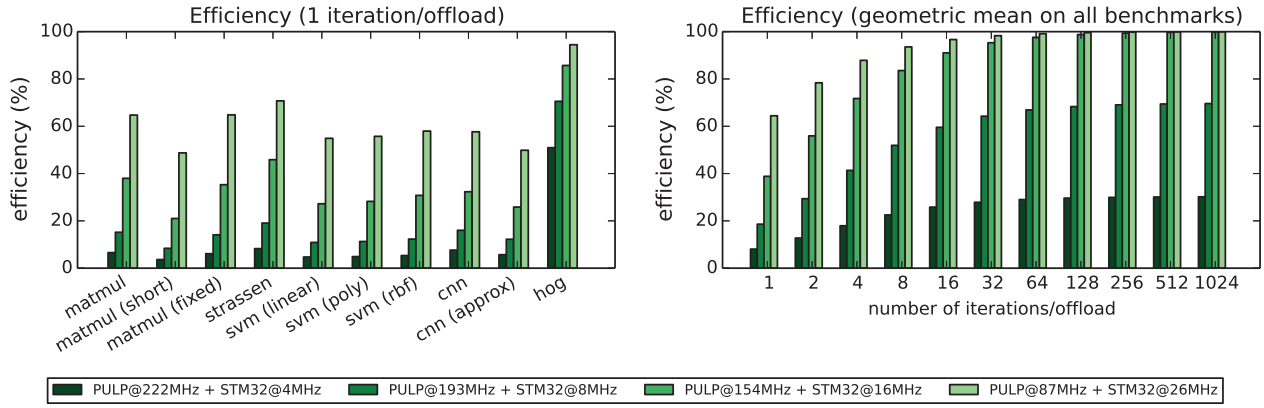
In the case of an embedded system, one is not typically interested in the best absolute possible performance, but rather in the best performance achievable in a given power envelope. To demonstrate that heterogeneous acceleration is an effective solution also at this level, we show what happens when we impose a constraint of 10 mW to the total power consumption, considering the MCU, PULP and the SPI link between the two. The baseline is given by clocking the *STM32-L476* MCU at 32 MHz. When the MCU is used at this frequency, there is no additional room for acceleration. If the frequency of the MCU is below 32 MHz, the available power can be used for the heterogeneous accelerator. As the MCU frequency is lowered, the power available for the accelerator is more, therefore it is possible to operate it at a higher frequency.

The plot in Figure 5a shows the pure PULP vs STM32 speedup over the baseline (STM32 at 32 MHz) in all combinations, allowing the accelerator to run at the maximum speed allowed by the available power envelope. Note that in this plot we do not yet consider the cost of the offload procedure itself. For better clarity, the plot also shows the performance gain that could be triggered by spending more than the allotted 10 mW raising the MCU frequency. The two sets of PULP and MCU bars are also annotated to show the number of RISC ops/cycle in each benchmark. Within this 10 mW budget, usage of the accelerator ideally allows significant speedups - as much as 60x in the case of the fastest benchmark (`strassen`), more than 25x for all the fixed point benchmarks and 20x for the worst-case benchmark (`hog`).

However, offloading computation from the MCU to PULP is not for free, in terms of both performance (latency) and energy. We have two limiting factors to take into consideration: the impact of the accelerator binary offload, and that of the input/output data transfer between the host MCU and the accelerator. Figure 5b takes into account this scenario, showing the efficiency loss due to this effect when we consider a single iteration of the benchmark (e.g. one frame/offload for the vision benchmarks), and how this efficiency can be recovered by increasing the number of benchmark iterations performed per each offload. To calculate throughput and power consumption of the coupling link we considered using the QSPI interface featured by the *STM32-L476* MCU. We can distinguish two scenarios: if the SPI link between the MCU and the accelerator is fast enough, the computation time dominates and full efficiency can be reached after as few as 32 iterations; this is the case of the two configurations in which the STM32 is fastest (16MHz and 26 MHz, respectively). Conversely, if the bandwidth of the SPI link is too low, the efficiency reaches a plateau. The maximum speedup can thus be reached by clocking the MCU at the minimum frequency at which the SPI does not become a bottleneck.



(a) Speedup without offload achievable vs STM32@32MHz within a 10mW power envelope (annotated with RISC ops/cycle).



(b) Efficiency w.r.t. ideal speedup when scaling the number of benchmark iterations per offload.

Figure 5: Speedup achievable within a total 10mW power envelope.

Clearly this could be considered a worst-case situation, where the entire cost for the offload is paid for a tiny accelerated piece of computation. In practical cases, more often than not the same computation is iterated over a large set of data chunks. As a consequence, the cost for code offloading is quickly amortized. Data still has to be brought in and out of the accelerator memory, but traditional double buffering schemes can be implemented to overlap data transfers with useful computation, which is the case we show in the rightmost plot. We observe that the limited efficiency that can be observed for some operating points in Figure 5b is strictly dependent from the very low frequency at which the MCU is clocked in those points, which on turn severely limits the SPI frequency and throughput. In a practical situation, even when performing most of the computation in those operating points the MCU frequency might be raised for enough time to efficiently perform the data exchange, easily overcoming this limitation. We refer to Section V for an extended discussion on this topic.

V. Discussion

Heterogeneous acceleration in the IoT design space answers to a different set of requirements with respect to other domains. Although energy efficiency is extremely important, absolute power consumption is also a first-class citizen; IoT nodes are severely constrained in terms of cost and power delivery, which is usually implemented with small batteries and/or harvesters [1]. For this reason, we chose to present our results in terms of total speedup subject to an overall power budget, rather than maximum achievable speedup or generic energy efficiency improvement.

In our model, we chose to avoid the integration of the MCU and of the accelerator in the same chip; instead, we propose to use an accelerator built in 28nm FD-SOI technology and an off-the-shelf MCU fabricated in a more "conservative" technology node as the host. This choice answers to cost considerations specific of the ULP domain. The accelerator absolutely needs a high level of integration to achieve the high energy-efficiency that can be "spent" to provide a high level of speedup. Conversely, MCUs are typically fabricated with a different set of goals, such as very low mask cost and fast turn-around time due to the need to produce a great number of models differing by interfaces, memory size, etc. To amortize the non-recurrent engineering costs of the higher density technology, it is necessary to produce the accelerators in high volume; therefore, the most sensible approach is to make them able to couple with the highest possible number of microcontrollers on the market. In the model we propose, this is achieved using an SPI interface, which is available on the overwhelming majority of MCU platforms and allows relatively easy and cheap integration of the host and accelerator in a system-on-board. It also has the significant advantage of being fully retrocompatible with the programming legacy, as it builds on existing MCU programming models without disrupting them.

The model presented in Section III-A, and whose performance/efficiency results in a practical embodiment are discussed in Section IV, provides a starting point for the development of sub-10mW accelerators. As we observed in Section IV-B

regarding the results shown in Figure 5b, the bottleneck of the SPI coupling link between the host and the accelerator can be lifted by temporarily raising the MCU frequency when performing a data transfer. It would be more desirable to have a low-power, high-throughput SPI link that is not tied to the MCU core frequency and that completely removes the bottleneck. Bringing this further, a possible variation on the model we propose in Figure 1 is to bring data from the sensor directly to the internal memory of the accelerator. This requires a dedicated (and more expensive) interface between the sensor and the accelerator, but it also reduces the pressure on the coupling link in terms of throughput, while it can still be used for pipeline cooperation between accelerator and host.

Finally, an interesting point to raise is that of the heterogeneity of tasks in the ULP domain. While in this work we mainly concentrate on a single task that is performed either on the host or on the accelerator, we modeled our power budget to allow for an additional, separate task to be performed on the host at the same time. This would allow for even more complex functionality to be performed in the sub-10mW space, taking advantage of the relative strengths of the host and the accelerator.

VI. Conclusions

In this work we propose the heterogeneous accelerator model as a methodology to enhance performance and energy efficiency of computation-intensive kernels in embedded scenarios with strong requirements on low power consumption; we also show a particular embodiment of this paradigm by coupling a STM32 microcontroller with PULP, a programmable ULP parallel accelerator. We present a simple offload programming model, designed as a subset of the widespread OpenMP specifications, that abstracts the low-level details of communicating code and data from the STM32 to PULP via a SPI interface, thus significantly improving the programmability of the system. Our results show that a speedup of more than an order-of-magnitude is achievable in a ULP setting, without compromising the platform's programmability.

Acknowledgements

This work was funded by the European Union *EuroCPS H2020-ICT-2014* Grant (G.A. 644090) and by the *MicroLearn: Micropower Deep Learning* Swiss NSF project. We also thank STMicroelectronics for granting access to the FDSOI 28nm technology libraries.

References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, pp. 1497–1516, Sept. 2012.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, pp. 1645–1660, Sept. 2013.

[3] D. Mellis, M. Banzi, D. Cuartielles, and T. Igoe, "Arduino: An Open Electronics Prototyping Platform," in *Proceedings of the ACM Conference on Human Factors in Computing*, vol. 2007, 2007.

[4] "Ambiq Apollo Data Brief."

[5] H. Ghasemzadeh and R. Jafari, "Ultra Low-power Signal Processing in Wearable Monitoring Systems: A Tiered Screening Architecture with Optimal Bit Resolution," *ACM Trans. Embed. Comput. Syst.*, vol. 13, pp. 9:1–9:23, Sept. 2013.

[6] D. Rossi, A. Pullini, M. Gautschi, I. Loi, F. K. Gurkaynak, P. Flatresse, and L. Benini, "A -1.8V to 0.9V Body Bias, 60 GOPS/W 4-Core Cluster in Low-Power 28nm UTBB FD-SOI Technology," in *Proceedings of the 2015 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (to appear)*.

[7] "STMicroelectronics STM32F407xx Datasheet."

[8] "STMicroelectronics STM32F446xx Datasheet."

[9] "NXP LPC185x/3x/2x/1x Datasheet."

[10] "SiliconLabs EFM32 Microcontroller."

[11] "Texas Instruments MSP430 Low-Power MCUs."

[12] "STMicroelectronics STM32L476xx Datasheet."

[13] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, vol. 98, pp. 253–266, Feb. 2010.

[14] N. Ickes, Y. Sinangil, F. Pappalardo, E. Guidetti, and A. P. Chandrakasan, "A 10 pJ/cycle ultra-low-voltage 32-bit microprocessor system-on-chip," in *2011 Proceedings of the ESSCIRC (ESSCIRC)*, pp. 159–162, IEEE, Sept. 2011.

[15] D. Bol, J. De Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J.-D. Legat, "SleepWalker: A 25-MHz 0.4-V Sub-mm2 7-uW/MHz Microcontroller in 65-nm LP/GP CMOS for Low-Carbon Wireless Sensor Nodes," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 20–32, Jan. 2013.

[16] F. Botman, J. D. Vos, S. Bernard, F. Stas, J.-D. Legat, and D. Bol, "Bellevue : a 50MHz Variable-Width SIMD 32bit Microcontroller at 0 . 37V for Processing-Intensive Wireless Sensor Nodes," in *Proceedings of 2014 IEEE Symposium on Circuits and Systems*, pp. 1207–1210, 2014.

[17] V. K. Singhal, V. Menezes, S. Chakravarthy, and M. Mehendale, "A 10.5 uA/MHz at 16MHz Single-Cycle Non-Volatile Memory Access Microcontroller with Full State Retention at 108nA in a 90nm Process," in *Solid- State Circuits Conference - (ISSCC), 2015 IEEE International*, pp. 1–3, Feb. 2015.

[18] D. Fick, R. G. Dreslinski, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, T. Mudge, D. Blaauw, and D. Sylvester, "Centip3De: A Cluster-Based NTC Architecture With 64 ARM Cortex-M3 Cores in 3D Stacked 130 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 104–117, Jan. 2013.

[19] S. Seo, R. G. Dreslinski, M. Woh, C. Chakrabarti, S. Mahlke, and T. Mudge, "Diet SODA: A Power-Efficient Processor for Digital Cameras," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design - ISLPED '10*, (New York, New York, USA), p. 79, ACM Press, 2010.

[20] A. Y. Dogan, D. Atienza, A. Burg, I. Loi, and L. Benini, "Power/performance exploration of single-core and multi-core processor approaches for biomedical signal processing," *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*, pp. 102–111, 2011.

[21] "NXP LPC54100 Datasheet."

[22] "Freescale MC9S12XDP512 Datasheet."

[23] K.-M. Lim, S.-W. Jeong, Y.-C. Kim, and H. S. Yang, "CalmRISC^{TM}: a low power microcontroller with efficient coprocessor interface," *Microprocessors and Microsystems*, vol. 25, pp. 247–261, Aug. 2001.

[24] A. Hodjat, D. Hwang, L. Batina, and I. Verbauwhede, "A hyperelliptic curve crypto coprocessor for an 8051 microcontroller," in *IEEE Workshop on Signal Processing Systems Design and Implementation, 2005*, pp. 93–98, Nov. 2005.

[25] T. Fujita, T. Tanaka, K. Sonoda, K. Kanda, and K. Maenaka, "Ultra Low Power ASIC for R-R Interval Extraction on Wearable Health Monitoring System," in *2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3780–3783, Oct. 2013.

[26] N. Ozaki, Y. Yoshihiro, Y. Saito, D. Ikebuchi, M. Kimura, H. Amano, H. Nakamura, K. Usami, M. Namiki, and M. Kondo, "Cool Mega-Array: A highly energy efficient reconfigurable accelerator," in *2011 International Conference on Field-Programmable Technology (FPT)*, pp. 1–8, Dec. 2011.

[27] A. Marongiu, A. Capotondi, G. Tagliavini, and L. Benini, "Simplifying Many-Core-Based Heterogeneous SoC Programming With Offload Directives," *IEEE Transactions on Industrial Informatics*, vol. 11, pp. 957–967, Aug. 2015.

[28] *OpenRISC 1000 Architecture Manual*. 2012.

[29] M. Gautschi, M. Scandale, A. Traber, A. Pullini, A. Di Federico, M. Beretta, G. Agosta, and L. Benini, "Tailoring Instruction-Set Extensions for an Ultra-Low Power Tightly-Coupled Cluster of OpenRISC Cores," in *Proceedings of VLSI-SOC 2015*, 2015.

[30] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A Fully-Synthesizable Single-Cycle Interconnection Network for Shared-L1 Processor Clusters," in *2011 Design, Automation & Test in Europe*, pp. 1–6, IEEE, Mar. 2011.

[31] D. Rossi, I. Loi, G. Haugou, and L. Benini, "Ultra-Low-Latency Lightweight DMA for Tightly Coupled Multi-Core Clusters," in *Proceedings of the 11th ACM Conference on Computing Frontiers - CF '14*, (New York, New York, USA), pp. 1–10, ACM Press, 2014.

[32] I. Miro-Panades, E. Beigné, Y. Thonnart, L. Alacoque, P. Vivet, S. Lesecq, D. Puschini, A. Molnos, F. Thabet, B. Tain, K. B. Chehida, S. Engels, R. Wilson, and D. Fuin, "A Fine-Grain Variation-Aware Dynamic Vdd-Hopping AVFS Architecture on a 32 nm GALS MPSoC," *IEEE Journal of Solid-State Circuits*, vol. 49, pp. 1475–1486, July 2014.

[33] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, May 2011.

[34] F. Conti, A. Pullini, and L. Benini, "Brain-inspired Classroom Occupancy Monitoring on a Low-Power Mobile Platform," in *CVPR 2014 Workshops*, 2014.

[35] A. Vedaldi and B. Fulkerson, "VLFeat: An Open and Portable Library of Computer Vision Algorithms," in *Proceedings of the International Conference on Multimedia*, MM '10, (New York, NY, USA), pp. 1469–1472, ACM, 2010.