Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays

VAHID LARI, SHRavan MUDDASAni, SRINIVAS BOppu, FRANK HANNIG, MORITZ SCHMID, and JÜRGEN TEICH, University of Erlangen-Nuremberg

We present a self-adaptive hierarchical power management technique for massively parallel processor architectures, supporting a new resource-aware parallel computing paradigm called invasive computing. Here, an application can dynamically claim, execute, and release the resources in three phases: resource acquisition (invade), program loading/configuration and execution (infect), and release (retreat). Resource invasion is governed by dedicated decentralized hardware controllers, called invasion controllers (iCtrls), which are integrated into each processing element (PE). Several invasion strategies for claiming linearly connected or rectangular regions of processing resources are implemented. The key idea is to exploit the decentralized resource management inherent to invasive computing for power savings by enabling applications themselves to control the power for processing resources and invasion controllers using a hierarchical power-gating approach. We propose analytical models for estimating various components of energy consumption for faster design space exploration and compare them with the results obtained from a cycle-accurate C++ simulator of the processor array. In order to find optimal design trade-offs, various parameters like (a) energy consumption, (b) hardware cost, and (c) timing overheads are compared for different sizes of power domains. Experimental results show significant energy savings (up to 73%) for selected characteristic algorithms and different resource utilizations. In addition, we demonstrate the accuracy of our proposed analytical model. Here, estimation errors less than 3.6% can be reported.

Categories and Subject Descriptors: C.1.2 [Processor Architectures]: Multiple Data Stream Architectures (Multiprocessors)—Array and vector processors; C.1.3 [Processor Architectures]: Other Architecture Styles—Adaptable architectures

General Terms: Design, Experimentation, Management, Performance

Additional Key Words and Phrases: Adaptive power optimization, parallel computing, resource awareness, runtime resource management, timing overhead minimization

ACM Reference Format:
DOI = 10.1145/2390191.2390193 http://doi.acm.org/10.1145/2390191.2390193

1. INTRODUCTION

With CMOS process technology reaching the physical limits of the fundamental device as well as ever-increasing need for more functionality and computational power, there is a strong trend towards research in multi-core and even many-core system-on-chip...
architectures. Such architectures enable a high degree of parallelism and offer high performance at the same power level along with flexibility by supporting multiple small computations compared to single core processors. Examples of such architectures include IBM’s Power7 chip [Kalla et al. 2010], comprising eight processor cores, each having 12 execution units with four-way simultaneous multithreading, the Intel’s Single-Chip Cloud Computer (SCC) with 48 cores on a single chip [Howard et al. 2010], Picochip’s PC-200 series [Duller et al. 2003] with 200–300 cores per device, Tilera’s TILE-Gx processor family with up to 100 64-bit VLIW processor cores, and the Am2045 [Butts 2007], a massively parallel processor array from Ambric having 336 cores. This trend has become even more aggressive toward having thousands of cores on a single chip, such as in Adapteva’s Epiphany processor series which contains up to 4,096 cores on a single chip.1

Architectures such as NEC’s DRP [Motomura 2002], PACT XPP [Baumgarte et al. 2003], or ADRES [Bouwens et al. 2008] are processor arrays that can be switched between multiple contexts by runtime reconfigurations. Mapping approaches for all the aforementioned architectures have in common that they require static knowledge about the number of available resources at compile time. Furthermore, these architectures are controlled centrally and often provide no mechanisms for managing the utilization of the computing resources.

However, increasing the number of processing elements—especially in the case of many-core architectures—induces several design challenges to the system, such as resource management, fault-tolerance, cost, performance, and power issues, as well as communication topology, memory architecture, and many others.

Managing and supervising such an amount of resources, if performed completely centralized, may become a major system performance bottleneck, and thus, current approaches may not scale any longer. Therefore, there is a strong trend toward dynamic exploitation of the available level of parallelism in many-core architectures based on the application requirements. For instance, in the TRIPS project [Sankaralingam et al. 2006], an array of small processors is used for the flexible and dynamic allocation of resources to different types of concurrency, ranging from running a single thread on a logical processor composed of many distributed cores to running many threads on separate physical cores. In the CAPSULE project [Palatin et al. 2006], the authors describe a component-based programming paradigm combined with hardware support for processors with simultaneous multithreading in order to handle the parallelism in irregular programs.

In Teich [2008] proposed a novel paradigm called invasive computing for designing and programming future massively parallel computing systems, such as a heterogeneous multiprocessor system-on-a-chip (MPSoC). The main idea and novelty of invasive computing is to introduce resource-aware programming support in the sense that a given application gets the ability to explore and dynamically spread its computations to other processors in a phase called invasion, and then to execute code segments with a high degree of parallelism based on the region of claimed resources on the MPSoC. Afterward, once the application terminates or if the degree of parallelism should be decreased, it may enter a retreat phase, deallocate resources, and resume execution again, for example, sequentially on a single processor.

In today’s CMOS process technology, it is indeed possible to integrate hundreds of processing elements on a single die. However, power consumption is considered a major challenge for such chips. There is an increasing need for energy efficiency because of limited battery capacity, energy costs, environmental impact, and system reliability. Thus, power has become one of the most important design parameters.


In on-chip massively parallel processing arrays, such as tightly-coupled processor arrays (TCPAs) [Kissler et al. 2006], applications may not exploits the full extent of available parallelism, hence there can be many idle processing elements (PEs). Although idle PEs may not consume dynamic power, they still contribute significantly to the total power consumption because of their clock-tree and static power consumption.

In this work, we present a novel hierarchical and adaptive technique for powering off the idle components of tightly-coupled processor arrays; see Figure 1 as an example. Each PE of this architecture consists of a processing unit (PU) and a dedicated hardware controller, called the invasion controller (iCtrl), which performs resource exploration, acquisition, and release in a distributed manner. We exploit invasion requests to shut down and wake up processors only during phases of invasion initiated by the applications themselves. To take advantage of these invasion phases, two different kinds of power domains are considered: processing unit power domains and invasion controller power domains. These domains are controlled hierarchically based on the system utilization, which in turn is controlled by the invasion controllers. Furthermore, we evaluate the effects of the size of invasion controller power domains on the overall system energy consumption, hardware cost, and invasion latency, that is, how fast we can acquire the resources and release them.

The remainder of the article is organized as follows. A brief overview of related work is given in the next section. Section 2.1 describes the architecture of a TCPA alongside an explanation of various design components. This is followed by Section 2.2, where our distributed resource exploration methodology is explained. In Section 3, the proposed adaptive power architecture concept is discussed, which is augmented with implementation details in Section 3.1. Mathematical models for estimating dynamic and static energy are given in Section 4. Section 5 presents various experiments for architecture alternatives by grouping several PEs into different power domains and
the results comparing area, invasion latency, and power from the proposed approach. Finally, Section 6 concludes the article and gives an outlook of future research.

2. RELATED WORK
For designs below 90 nm process technology, static and clock-tree power account for approximately 60% of the total power [Kao et al. 2002; Homayoun et al. 2011].

There are many techniques proposed in the literature for minimizing clock-tree and static power of idle components in the design. Kissler et al. [2008] show a 35% reduction in clock-tree power with an RTL-based clock-gating approach on massively parallel processor arrays. Wu et al. [2000] achieve a similar reduction in clock-tree power. Techniques involving control of the power supply yield major savings in power but have a bigger impact on the design methodology and implementation [Hailin et al. 2005]. With a reconfiguration-controlled centralized approach for power gating, Kissler et al. [2011] achieved 60% savings in leakage power and for coarse-grained reconfigurable processor arrays, Saito et al. [2008] achieved 48% savings in leakage power. The key differentiator for clock and power gating techniques is in predicting the idle states dynamically, which in most cases depends on the architecture and the applications. In this article, we propose exploiting the advantages of invasive computing in order to implement a decentralized, adaptive, and hierarchical power gating technique for tightly-coupled processor arrays. Here, the applications dynamically derive the power gating signals locally at every processing element, waking up processors implicitly at invade time and shutting them down automatically at retreat time.

2.1. Tightly-Coupled Processor Arrays
A tightly-coupled processor array (TCPA), as shown in Figure 1, is a massively parallel architecture consisting of an array of tightly-coupled PEs, each having a very long instruction word (VLIW) structure [Kissler et al. 2006]. Each PE may be parameterized for a range of functionality at synthesis time, such as adding, multiplying, shifting, and logical operations that are implemented as separate functional units which can work in parallel. The amount of instruction memory and control overhead of PEs is kept as small as possible. Consequently, interrupts, exceptions, data, or instruction caches, as well as multithreaded execution, are not supported by PEs.

Tight coupling and cycle-based communication are established by point-to-point connections among PEs. Also, different interconnection topologies can be implemented by the reconfiguration of interconnection routers at runtime.

As the processing elements are tightly coupled, they do not have direct access to the global memory. Data transfer to and from the array are performed through the PEs at the borders which are connected to the banks of surrounding memory buffers. These buffers can be configured to either work as simple FIFO buffers or as RAM-based addressable memory banks (Figure 1). This architecture can be connected to a bus or a network-on-a-chip system to work as a hardware accelerator in an MPSoC for executing computationally intensive applications. The admission of an application on the processor array, communication network initialization, and processor array reconfiguration is managed by a RISC processor called the Reconfiguration and Communication Processor (see also Figure 1). Each PE of this architecture is further equipped with a hardware component called the invasion controller (iCtrl), giving it the capability to acquire, reserve, and then release the PEs in its neighborhood. Such capabilities are discussed in the context of invasive computing. This concept as well as its integration into the TCPA architecture is explained in the next section.
2.2. Invasive Computing

Invasive computing is a new resource-aware parallel computing paradigm that has been conceived in order to overcome the resource scalability problems in future many-core architectures [Teich 2008; Teich et al. 2011]. Using this paradigm, each application gains the ability to explore and reserve (invade) available resources in a specific neighborhood, copy its configuration code (infect) to such a set of captured resources, and then to execute the given program in parallel on the employed resources. After finishing a phase of execution, the application may free its previously occupied resources by performing a release operation (retreat) [Teich et al. 2011; Hannig et al. 2011].

This process takes place in a distributed fashion, starting from an initial element of asking its neighboring resources about their availability of sending so called INV signals. Each INV signal includes information about shape and the size of the region requested to be captured. This process is continued by the neighboring resources in a cycle-by-cycle fashion until the required number of resources is reserved. As soon as the application's requirements (the number of needed PEs) are fulfilled, the invaded PEs start sending back confirmation signals, which are called INV_CLAIM, indicating information about the invaded PEs, such as the number of invaded PEs and their location. This process is performed in the reverse direction of the invasion, starting from the last PE in the invaded domain to the PE that initiated the invasion. After subsequently loading the configuration into the captured PEs and completing the execution of the application, the initiator PE will issue a retreat signal (RET) to its captured neighbors. Similar to an INV signal, a RET signal may include information about the size of the region that is requested for release. An invaded region could be released completely or partially. This signal is locally propagated through all captured PEs, following the same path paved by the invade signal, until all PEs captured by the application are signaled for release, which is again confirmed by a wave of signals (RET_CLAIM) from the last PE to the initiator PE.

It should be mentioned that the task of choosing a suitable place for starting an invasion is handled by the Reconfiguration and Communication Processor (see Figure 1). This processor receives application requests, then it selects a suitable PE from some dedicated PEs—called invade initiators—to start an invasion in the array and sends an INV signal to the selected PE.

Based on many application requirements for 1D and 2D images, signal processing, and other compute-intensive loop specifications, Lari et al. [2011] have proposed two major invasion strategies: linear invasion and rectangular invasion. Linear invasion, which results in capturing a linearly connected chain of PEs, is suitable for types of one-dimensional digital signal processing applications, such as FIR filters. Rectangular invasions reserve rectangular regions of connected PEs and are suitable for two-dimensional image processing applications, such as algorithms for edge detection, image filtering, and optical flow. These two strategies are depicted in Figure 1, where a region is captured by a rectangular invasion in the upper-left corner of the array (green), and a linear region is captured at the bottom (yellow).

In order to support the propagation of invasion signals, each processing element of a many-core architecture must be equipped with an invasion controller (iCtrl) (see Figure 1). Lari et al. [2011] propose two different designs of such controllers; an FSM-based design, which implements the mentioned invasion strategies in hardware, and a programmable controller that executes software-based invasion strategies. In this work, due to timing overhead and power consumption constraints, we use the FSM-based design. This controller is able to send each invasion-related signal in a single cycle; therefore, on average, it takes about two cycles to invade/retreat each PE (one cycle for the INV/RET signals and one cycle for the INV/RET_CLAIM signals).
Lari et al. [2011] also show that by using such an invasion controller, the process of resource exploration, reservation, and retreat could be performed up to 45 times faster than a centralized software approach. We have integrated the controller into each processing element of the TCPA, as shown in Figure 1. The resulting architecture minimizes the overhead of control flow, memory access, as well as data transmissions, indeed, by inheriting the dynamism and self-adaptiveness of invasive TCPAs, energy consumption can be optimized by dynamically powering off the idle regions in the array at retreat time. This feature is especially helpful when the architecture is used as a hardware accelerator inside portable devices, where battery life is critical. The next section explains our concepts for dynamically adapting the powered regions to just the level of utilization.

3. ADAPTIVE POWER MANAGEMENT

In the following, our goal is to study how the principle of invasion also enables an inherent way of saving power. Our resource-aware computing methodology shows its importance when targeting many-core architectures consisting of tens to thousands of processing elements. Such a great number of computational resources increases the level of system parallelism, but on the other side, may also cause a high level of power consumption. One traditional way to decrease the overall system power dissipation is to decrease the amount of static power by powering off unused resources. With invasive computing, a system can tune its utilization level to the required level of parallelism. In this work, we benefit from the capability to adapt the system power consumption automatically with its utilization level. This is realized by switching on only the invaded PEs and keeping the idle PEs disconnected from the power supply. As shown in Figure 1, each PE is composed of two major parts: a processing unit (PU) that includes the computational core of the PE and an invasion controller (iCtrl) that processes the invasion requests. In order to achieve the highest power savings, we apply power switching capabilities to both parts, according to Figure 2(a). Each of these parts may operate either in powered on or powered off mode.

All PEs in the array start their operation in the powered off mode, that is, all processing units and invasion controllers are switched off. Now, once a PE receives an invade signal (INV), its invasion controller is first powered on, making it available for evaluating and reacting to the invasion request. This request may lead to the continuation of the invasion by sending an INV signal to a neighboring PE, which results in switching on its own invasion controller, as well. This process continues until the last required PE in the array is reached and the confirmation phase is started. At this stage, when a PE sends an INV_CLAIM signal to its predecessor PE, it switches off its invasion controller and switches on the processing unit, making the processing unit available for the application configuration and starting the application execution (infect phase). When the application execution is finished, the PEs are released by the invasion controllers by issuing an RET signal, which is propagated in a similar way as in the invasion phase: First, the invasion controller of a PE that has received an RET signal is switched on, then the PE forwards the RET signal to the neighboring PE that it has invaded. At the same time, the PE switches off its processing unit, as it is not used by the application anymore. This wave of signaling continues through the last PE in the domain. It is then followed by a confirmation phase, which includes switching off the invasion controllers. Finally, all parts of the invaded PEs are switched off. Table I summarizes our power switching protocol for different phases of invasion (infect signal, confirmation by INV_CLAIM signals) and retreat (retreat signal, confirmation by RET_CLAIM signals).

According to this protocol, two different kinds of power domains may be distinguished—the processing unit and invasion controller power domains. As shown in Table II, although the size of the invasion controller is rather small, it imposes a
lot of implementation effort and hardware overhead if each is placed in its own power domain. Therefore, in this work, we study the effects of grouping multiple invasion controllers in the same domain, as shown in Figure 2(b), which—from one side—may reduce hardware overhead but—from the other side—may decrease our control on switching the power of the individual components. The finer the granularity of the power control, the more power we may save. At the same time, grouping more invasion controllers together will reduce the timing overhead that is needed for power switching during both the invasion and retreat phase. Figure 2 shows different example architectures for grouping invasion controllers. In the following, we explain how our power state machine controls the power of the processing unit power domains and invasion controller power domains.

![Diagram](image)

**Fig. 2.** Different designs for grouping invasion controllers (iCtrl) in one power domain.

**Table I. The Power-Switching Protocol**

<table>
<thead>
<tr>
<th>Invasion Signal</th>
<th>Processing Unit</th>
<th>Invasion Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>no change</td>
<td>OFF → ON</td>
</tr>
<tr>
<td>INV_CLAIM</td>
<td>OFF → ON</td>
<td>ON → OFF</td>
</tr>
<tr>
<td>RET</td>
<td>ON → OFF</td>
<td>OFF → ON</td>
</tr>
<tr>
<td>RET_CLAIM</td>
<td>no change</td>
<td>ON → OFF</td>
</tr>
</tbody>
</table>

**Table II. Hardware Cost of Different System Components per PE in Terms of NAND2 Equivalent Gates**

<table>
<thead>
<tr>
<th></th>
<th>Processing Unit</th>
<th>Invasion Controller</th>
<th>Power Management Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAND2 eq. gates</td>
<td>8,915</td>
<td>732</td>
<td>160</td>
</tr>
</tbody>
</table>
3.1. Power Controller

Power gating of the PEs and invasion controllers is controlled by a power management unit (PMU), according to Figure 2. A PMU is needed for each processing unit domain and invasion controller domain. The PMU controls the power state of its associated domains according to the invasion status signals, such as INV, INV_CLAIM, RET, and RET_CLAIM signals (see Figure 3 for more details). The PMU includes different power state machines (power-FSM): one for a set of invasion controllers that are grouped in the same power domain and a number of power state machines for controlling the power status of the associated processing units (based on the size of the power domains). As an example, each PMU in Figure 2(a) includes a power-FSM for its associated iCtrl and one for the associated PU. However, in Figure 2(b), each PMU consists of four power-FSMs for controlling four PUs and a single power-FSM for controlling the iCtrl power domain, which contains four invasion controllers. At the physical design level, a coarse-grained power gating strategy is used, where the switch cells are arranged in a grid style across the entire area of a power domain. Switch cells are placed equidistantly in the cell rows, arranged in a column-wise form, and are connected as a daisy chain (see Figure 3).

Each power-FSM receives invasion-related signals as its inputs and gives out the following signals to the switch cell daisy chain: Power_en, Clock_en, reset, isolate, and Power_on. When the PMU decides to power on a domain, its power-FSM initially sets the Power_en signal high, which is connected to P_in of the first switch cell, which traverses through all the daisy-chained switch cells, turning on the power gradually to ensure a surge-free power-on/off sequence (see Figure 3). The PMU receives the acknowledgment of the power-on/off from the Power_on signal, which is connected to the P_out of the last switch cell. The switch cells are from a standard cell library, and the buffer between P_in and P_out of the switch cell is connected to the gated power supply. After a programmable number of clock cycles (to ensure that the power domain
is completely turned on), the reset signal is asserted, and the clock gating is removed by setting \( \text{Clock	extunderscore en} \) to high. At the end, isolation is turned off by setting the signal isolate to low. Similarly, when the PMU switches off the domain, the sequencing of the signals is reversed, with no reset involved during the process. In summary, we have the following.

—Initially, all the PEs and invasion controllers are in the power-off state.
—When the PMU receives the INV signal, the power state of the invasion controller is transitioned from the power-off state to the power-on state.
—The INV	extunderscore CLAIM signal transitions the state of the PE from the power-off state to the power-on state and the invasion controller from the power-on state to the power-off state.
—The RET signal transitions the state of the invasion controller from power-off to power-on.
—When the RET	extunderscore CLAIM is received, the states of the PE and the invasion controller are changed from power-on to power-off.

4. ENERGY ESTIMATION MODEL
As previously mentioned, the size of the invasion controller power domains may affect the invasion latency as well as the energy consumption of the system. In this section, we present mathematical models in order to estimate the total energy consumption with respect to the size of the invasion controller power domains. In order to simplify the models, we consider only scenarios where a single application is mapped to the system; however, the experimental results (presented in Section 5) include test bench scenarios with multiple concurrent applications. Our mathematical models are presented for a processor array of size \( N_{\text{row}} \times N_{\text{col}} \), where \( N_{\text{row}} \) and \( N_{\text{col}} \) refer to the number of PE rows and columns in the array, respectively. The invasion controller power domains are assumed to be rectangular, where \( N_{i\text{Ctrl}_{\text{row}}} \) and \( N_{i\text{Ctrl}_{\text{col}}} \) are the number of rows and columns of invasion controllers in the same power domains. Consequently, the size of each invasion controller power domain is \( N_{i\text{Ctrl}_{\text{size}}} = N_{i\text{Ctrl}_{\text{row}}} \times N_{i\text{Ctrl}_{\text{col}}} \). Also, the following power switching timing parameters are considered: \( t_{i\text{Ctrl}_{\text{ON}}} \) and \( t_{i\text{Ctrl}_{\text{OFF}}} \) are the time delays for switching on/off an \( i\text{Ctrl} \) power domain; \( t_{\text{PU}_{\text{ON}}} \) and \( t_{\text{PU}_{\text{OFF}}} \) are the delays for switching on/off a PU power domain. Moreover, \( t_{\text{INV}} \) is the time needed for sending an INV signal to a PE; \( t_{\text{RET}} \) is the time needed for sending an RET signal to a PE; \( t_{\text{INV}}\text{CLAIM} \) and \( t_{\text{RET}}\text{CLAIM} \) are the time delays needed for sending the INV	extunderscore CLAIM and RET	extunderscore CLAIM signals, respectively. In the following section, these design parameters are used to evaluate the energy consumption of a TCPA for both linear and rectangular invasions.

4.1. Energy Model for Linear Invasions
As explained in Section 2.2, linear invasions are performed in a meander-walk fashion. A linear invasion is pursued in a row until reaching a turn point, that is, the boundary of an invasion or a PE that is already invaded, then it is continued in the upper or lower row in the opposite direction. The turn point \( (N_{\text{tp}}) \) is the maximum number of consecutive PEs in the same row that can be invaded, and it is defined to reserve the resources in a packed rectangular convex box. It also may be defined as equal to \( N_{\text{row}} \), which means that the invasion in a row continues until reaching the boundary of an array. By invading all the PEs in a row until the turn point, \( N_{i\text{Ctrl}_{\text{tp}}} = \left\lceil \frac{N_{\text{row}}}{N_{i\text{Ctrl}_{\text{col}}}} \right\rceil \) power domains have to be turned on. When an \( i\text{Ctrl} \) in an already powered-off \( i\text{Ctrl} \) domain receives an invasion-related signal, all controllers within the domain are powered on. These controllers are kept in the power-on state until all the invaded invasion controllers in the domain send their confirmation signals (INV or RET	extunderscore CLAIM signals). Figure 4(b) shows this fact, in which the power-on delays appear only when the first PE in an \( i\text{Ctrl} \) domain
power domain is invaded. If a PE is processing an invasion-related signal, it consumes both static and dynamic energy, but if a PE is powered on and processes no invasion signal, it just consumes static energy. The dynamic and static power consumption of invasion controllers are $P_{d\text{-iCtrl}}$ and $P_{s\text{-iCtrl}}$, and in the case of PUs, $P_{d\text{-PU}}$ and $P_{s\text{-PU}}$, respectively. In the following section, we present mathematical models for computing the energy consumption of invasion controllers and processing units.
Energy consumption of invasion controllers. The power switching behavior of invasion controllers follows the same protocol in both invade time and retreat time (Table I). Therefore, their energy consumption in both phases is the same. As explained in Section 2.2, each invasion (or retreat) consists of two subphases: an invade (retreat) signal propagation phase and a confirmation phase. These two phases are depicted in Figure 4, where the iCtrl power domains are first powered on sequentially, and then, in the confirmation phase, the power domains are powered off, again, in the reverse order sequentially.

According to Figure 4, during the invasion signal propagation phase, the invasion latency consists of two types of components: non-delayed parts in which the signals are propagated without any power-on delay within the already turned-on power domains, and delayed parts in which an invasion power domain is turned on in each of them.

The number of such delayed parts depends on the number of iCtrl power domains that are powered on during an invasion. In order to calculate the number of powered-on iCtrl power domains, we need to calculate the number of rows of PEs that are invaded. This is due to the row-wise moving behavior of the linear invasions. For \( N_{inv} \) invaded PEs and a turn point equal to \( N_{tp} \), the number of rows of PEs (yellow blocks in Figure 4)(a) that are invaded is calculated as

\[
N_{PE rows} = \left\lceil \frac{N_{inv}}{N_{tp}} \right\rceil.
\]

As Figure 4(a) shows, a row of iCtrl power domains (blue blocks) may contain a number of PE rows (yellow blocks), based on \( N_{iCtrl row} \). The number of blue blocks (rows of iCtrl power domains) participating in an invasion is derived by \( N_{iCtrl ON row} = \left\lceil \frac{N_{PE rows}}{N_{iCtrl row}} \right\rceil \). The number of iCtrl power domains turned on (\( N_{iCtrl ON} \)) is given in Equation (1). When invading PEs within the blue blocks, if the number of invaded PEs in the block exceeds the \( N_{tp} \), then it implies that multiple rows of PEs (yellow blocks) are captured, meaning all iCtrl power domains within that blue block should be powered on (the second part of Equation (1)). Otherwise, if less than \( N_{tp} \) PEs are invaded within a specific blue block, then some power domains may remain in the power-off state (the first part of Equation (1)). This is shown in Figure 4(a), where in the second blue block, the last iCtrl power domain remains powered off.

\[
N_{iCtrl ON} = \begin{cases} 
(N_{iCtrl ON row} - 1) \times N_{iCtrl,tp} + 
\frac{N_{inv} - (N_{iCtrl ON row} - 1) \times N_{iCtrl, row}}{N_{iCtrl, row}} 
\end{cases} \text{if } (N_{PE rows} \mod N_{iCtrl row}) = 1,
\]

\[
N_{iCtrl ON row} \times N_{iCtrl, tp} 
\text{else.}
\]

Now we can calculate the dynamic and static energy consumption for both phases of an invasion (signal propagation and confirmation). As explained, the INV (RET) signal propagation phase consists of two components: delayed and non-delayed parts. The total area of delayed parts (shown in gray and marked with an (A) in Figure 5) is calculated as the sum of the \( N_{inv} \) first natural numbers, weighted by the INV signal sending delay, that is, \( \sum_{n=1}^{N_{inv}} n \times t_{INV} \). The total area of the non-delayed parts is the sum of the area of orange blocks in Figure 5 (marked with (B)). The height of each component is equal to the number of powered-on invasion controllers, and its width is equal to the iCtrl power-on delay (\( t_{iCtrl, ON} \)). We can observe a repetitive pattern by increasing the area of (B) components. By switching on each iCtrl power domain within a row of iCtrl power domains (blue blocks), there is a partial height increment in the orange blocks equal to \( N_{iCtrl row} \). This can be observed in Figure 4(a), where multiple green blocks are crossed invading the first yellow block. This intra-row, domain-by-domain area increment is added to the number of iCtrl power domains that are already powered on.

Equation (2) calculates the area of components (A) and (B), and Equation (3) gives the corresponding dynamic energy consumption of iCtrl power domains within the INV
(A) components: The non-delayed active invasion controllers in the invasion signal propagation phase
(B) components: The delayed active invasion controllers in the invasion signal propagation phase
(E) components: The active invasion controllers in the confirmation phase

Fig. 5. Dynamic energy consumption of the invasion controller.

(RET) signal propagation phase ($E_{d,iCtrl1}^{(1)}$). $P_{d,iCtrl}$ is assumed to be the dynamic power consumption of each $iCtrl$.

$$\sigma_{A,B} = \sum_{n=1}^{N_{eic}} (n \times t_{INV}) + \sum_{n=1}^{N_{Ctrl\_row}-1} \left( \left\lfloor \frac{n}{N_{iCtrl\_col}} \right\rfloor \times N_{iCtrl\_row} \times N_{ip} \right) + \left( n - \left\lfloor \frac{n}{N_{iCtrl\_col}} \right\rfloor \times N_{iCtrl\_col} \right) \times t_{iCtrl\_ON}. \quad (2)$$

$$E_{d,iCtrl1}^{(1)} = \sigma_{A,B} \times P_{d,iCtrl}. \quad (3)$$

In case of static energy consumption, two major types of components are distinguishable in Figure 6: delayed parts ((C) components) and non-delayed parts ((D) components). The delayed parts correspond to the switching phases—mainly when the first PE of an $iCtrl$ power domain is invaded. Here, we observe a similar height increment pattern as that of the (B) components in the dynamic energy calculation. The height of the blocks is equal to the sum of all $iCtrl$ power domains that have been powered on. The width of these blocks is equal to the time needed for invading all PEs within the first row of an $iCtrl$ power domain plus the $iCtrl$ domain power-on delay. An exception may happen if the number of invaded PEs is less than a complete row of the $iCtrl$ power domain ($N_{iCtrl\_row}$). This situation may occur if $N_{PE\_rows} \mod N_{iCtrl\_row} = 1$, meaning that for the last blue block (the last row of the $iCtrl$ power domains), not all PEs available in a row are invaded. In such cases, the number of invaded PEs in the last $iCtrl$ power domain is derived by Equation (4). Equation (4) gives the area of (C) components, that is, the delayed parts in the propagation phase. This equation consists...
Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays

Fig. 6. Calculation of the static energy consumption of the invasion controller with separated energy calculation components.

The non-delayed parts ((D) components) in Figure 6 refer to invading the invasion controllers that are already powered on. This happens when \( N_{iCtrl_{row}} > 1 \) and the number of invaded PEs exceeds one row of PEs (i.e., multiple yellow blocks are invaded within a blue area in Figure 4(a)). The number of non-delayed blocks depends on the number of rows of iCtrl power domains (blue blocks) that are invaded (\( N_{iCtrl_{ON, row}} \)).

Figure 4(a) shows the corresponding invasion controllers to such non-delayed parts. We call such invasion controllers non-delayed iCtrls because in both phases of invasion signal propagation and confirmation, the signaling among them is performed without

\[
N_{inv_{iCtrl}} = \begin{cases} 
N_{inv} - \left( \frac{N_{inv}}{N_{iCtrl_{row}} - 1} \right) \times N_{iCtrl_{col}} & \text{if } N_{PE_{rows}} \mod N_{iCtrl_{row}} = 1; \\
N_{iCtrl_{col}} & \text{else.}
\end{cases}
\]  

(4)

\[
\sigma_C = N_{inv_{iCtrl}} \times t_{INV} \times N_{iCtrl_{size}} \times N_{iCtrl_{ON}} \\
+ \sum_{n=1}^{N_{iCtrl_{ON}-1}} n \times N_{iCtrl_{size}} \times (N_{iCtrl_{col}} \times t_{INV} + t_{iCtrl_{ON}}).
\]  

(5)
incorporation of any power switching delay. The number of such controllers within a blue area is equal to \((N_{iCtrl\_row} - 1) \times N_{tp}\). Equation (6) calculates the area of (D) components corresponding to the the static energy of the non-delayed parts of the INV (RET) signal propagation phase.

\[
\sigma_D = \sum_{n=1}^{N_{iCtrl\_row}} n \times N_{iCtrl\_size} \times N_{iCtrl\_tp} \times (N_{iCtrl\_row} - 1) \times N_{tp} \times t_{INV}.
\]  

(6)

Knowing the areas of the (C) and (D) components, the static energy consumption of \(i\)Ctrls in the signal propagation phase \(E_{iCtrl}^{(1)}\) is derived by Equation (7). \(P_{iCtrl}\) is assumed to be the static power consumption of each \(i\)Ctrl.

\[
E_{iCtrl}^{(1)} = (\sigma_C + \sigma_D) \times P_{iCtrl}.
\]  

(7)

For the dynamic energy consumption components in the confirmation phase \(E_{iCtrl}^{(2)}\) components), power switching delays do not contribute in the confirmation signaling delays (see Figure 5). The area of \(E\) components and the dynamic energy consumption for the confirmation phase are derived in Equation (8).

\[
\sigma_E = \sum_{n=1}^{N_{inv}} n \times t_{INV\_CLAIM}.
\]

\[
E_{iCtrl}^{(2)} = \sigma_E \times P_{iCtrl}.
\]  

(8)

As mentioned in Section 2.2, the confirmation phase starts from the last PE in the invaded domain, traversing the PEs in the reverse direction of the invade (retreat) propagation. An \(i\)Ctrl power domain is switched off when all invasion controllers within the power domain have sent back their invade (retreat) confirmation signals. As the confirmation phase is performed in the reverse direction of the invade (retreat) phase, a domain is powered off when the first PE that has been invaded (retreated) in the \(i\)Ctrl power domain confirms its invade (retreat). In Figure 6, three types of components for static energy consumption are observable: (F) components that correspond to the phase of confirmation when \(i\)Ctrl power domains are switched off one by one; (G) components that correspond to those phases of confirmation that non-delayed PEs (see Figure 4(a)) send their confirmation signals; and finally, (H) components that correspond to the timing delay of switching off the power domains. The height of (F) components is reduced at the rate of \(N_{iCtrl\_size}\) in each step. The total area of these components is given by Equation (9).

\[
\sigma_F = \sum_{n=1}^{N_{iCtrl\_ON} - 1} n \times N_{iCtrl\_size} \times N_{iCtrl\_col} \times t_{INV\_CLAIM}.
\]  

(9)

When non-delayed PEs (Figure 4(a)) send their confirmation signals, no power switching occurs. These phases are shown in Figure 6 as (G) components. The number of these blocks depends on the number of invaded rows of \(i\)Ctrl power domains (number of invaded blue blocks). The height of a (G) block is equal to the total number of switched-on invasion controllers, and its width is equal to the time needed to send confirmation signals by all non-delayed \(i\)Ctrls within the corresponding blue area. The area of such blocks is calculated by Equation (10).

\[
\sigma_G = \sum_{n=1}^{N_{iCtrl\_ON\_row} - 1} n \times N_{iCtrl\_row} \times N_{tp} \times (N_{iCtrl\_row} - 1) \times N_{tp} \times t_{INV\_CLAIM}.
\]  

(10)
Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays

Fig. 7. The PU’s switch-off sequence of PUs during an invade phase.

The power-off delay is equal to the time needed to turn off the first iCtrl power domain. As the whole process is pipelined, consequently, such a component contributes just once in a confirmation phase (see Figure 6). The area of an (H) component is given by Equation (11), and the total static energy consumption by invasion controllers in the confirmation phase \( E_{\text{s,iCtrl}} \) is given by Equation (12).

\[
\sigma_H = N_{\text{iCtrl, size}} \times N_{\text{iCtrl, ON}} \times t_{\text{iCtrl, OFF}}. \tag{11}
\]

\[
E_{\text{s,iCtrl}}^{(2)} = (\sigma_F + \sigma_G + \sigma_H) \times P_{\text{s,iCtrl}}. \tag{12}
\]

Equation (13) summarizes our evaluations for dynamic and static energy consumption of the invasion controllers.

\[
E_{\text{d,iCtrl}} = E_{\text{d,iCtrl}}^{(1)} + E_{\text{d,iCtrl}}^{(2)};
\]

\[
E_{\text{s,iCtrl}} = E_{\text{s,iCtrl}}^{(1)} + E_{\text{s,iCtrl}}^{(2)}. \tag{13}
\]

**Energy consumption of processing units.** According to Table I, during the invasion confirmation phase, the processing units are switched on, and they are switched off during the retreat signal propagation phase. The goal is to keep the PUs in the powered-on mode for the shortest time. Figures 7 and 8 show how the PUs are switched on and off during the invade and retreat phases, respectively. For simplification of the models, we assume that PUs start to operate in a fully active mode immediately when they are switched on. Consequently, they always consume both static and dynamic energy once they are powered on. As a result, the derived models for both static and dynamic energy are similar. Equation (14) calculates the number of components in the PU switch-on phase \((I)\) components), which is used to derive the static and dynamic energy consumption of PUs in this phase.

\[
\sigma_I = \sum_{n=1}^{N_{\text{PU}}} n \times t_{\text{INV,CLAIM}};
\]

\[
E_{\text{d,PU,ON}} = \sigma_I \times P_{\text{d,PU}};
\]

\[
E_{\text{s,PU,ON}} = \sigma_I \times P_{\text{s,PU}}. \tag{14}
\]
In case of the retreat phase, where the PUs are powered off, three types of components are observable (see Figure 8): power-off delay components (type (J)), PU power-off sequence that is done without incorporation of power switching delays (type (K)), and power switching delay components (type (L)). The (J) component occurs at the RET propagation phase, where invasion controllers are first switched on, then they switch off their associated PUs. Therefore, the invasion controller power-on delay, as well as the PU switch-off delays, are incorporated into this process. As the entire switch-off process takes place in a pipelined fashion, the power-off delay appears once (see Figure 8). The area of a (J) component is given by Equation (15).

\[ \sigma_J = N_{inv} \times (t_{\text{Ctrl,ON}} + t_{PU,OFF}). \]  

(15)

The area of (K) components is given by Equation (16).

\[ \sigma_K = \sum_{n=1}^{N_{inv}} n \times t_{\text{RET}}. \]  

(16)

Finally, (L) components appear, depending on the number of iCtrl power domains that are invaded. Equation (17) calculates the area of these components.

\[ \sigma_L = \sum_{n=1}^{N_{\text{iCtrl,DP}}-1} \left( N_{inv} - \left( n - \frac{n}{N_{\text{iCtrl,JP}}} \right) \times N_{iCtrl,JP} \right) \times N_{iCtrl,row} \]
\[ - \left( \frac{n}{N_{\text{iCtrl,JP}}} \right) \times N_{iCtrl,JP} \times N_{iCtrl,\text{size}} \times t_{\text{Ctrl,ON}}. \]  

(17)
The static (\(E_{dPU,OFF}\)) and dynamic (\(E_{dPU,OFF}\)) energy of PUs during the switching off phase is given by Equation (18):

\[
E_{d,PU,OFF} = (\sigma_J + \sigma_K + \sigma_L) \times P_{d,PU};
\]
\[
E_{s,PU,OFF} = (\sigma_J + \sigma_K + \sigma_L) \times P_{s,PU}.
\] (18)

The total static and dynamic consumption of PUs for an application running for \(t_{exe}\) cycles is simply calculated by summing up the energy consumption of the power switching phases plus the execution phase (Equation (19)).

\[
E_{d,PU} = E_{d,PU,ON} + N_{inv} \times P_{d,PU} \times t_{exe} + E_{d,PU,OFF};
\]
\[
E_{s,PU} = E_{s,PU,ON} + N_{inv} \times P_{s,PU} \times t_{exe} + E_{s,PU,OFF}.
\] (19)

### 4.2. Energy Model for Rectangular Invasions

As explained in Section 2.2, rectangular invasions result in capturing rectangular regions of PEs. Each rectangular invasion consists of a set of parallel vertical linear invasions, where each is initiated by the PEs in the first row of the rectangular region. Each vertical linear invasion proceeds in a single straight line of PEs (without any meander-like movement). This means that for an \(N_{inv,r} \times N_{inv,c}\) invaded rectangular region, the turning point is equal to \(N_{inv,c}\), where \(N_{inv,r}\) and \(N_{inv,c}\) correspond to the size of the rows and columns in the rectangular area, respectively. Figure 9 shows an example of performing a \(4 \times 4\) rectangular invasion over a \(4 \times 6\) TCPA. The linear invasions that are initiated within the same iCtrl power domain may exhibit the same power switching behavior. In an iCtrl power domain, the first vertical linear invasion acts as a pioneer invasion that reaches the border of the iCtrl power domain earlier than the other vertical linear invasions within the domain. This pioneer invasion triggers the power-on switching process in the next vertical iCtrl power domain. Consequently, when the other linear invasions reach this boundary, they need to wait for a shorter time. For the \(i\)th linear invasion within an iCtrl power domain, it is necessary to wait for \(t_{Ctrl,ON} - i \times t_{INV}\) cycles. In the case of the retreat phase, \(t_{INV}\) is replaced by \(t_{RET}\).

After turning on the next iCtrl power domain, all linear invasions continue with the same timing. In fact, the vertical boundary of the iCtrl power domain (the iCtrl power domain that contains the first row of the rectangular region) acts as a synchronizing barrier. If \(t_{Ctrl,ON} > i \times t_{INV}\), then the synchronization happens at the boundary of one of the upcoming iCtrl power domains. As we perform extremely fast invasions, where the timing overhead of invasion signaling is very low, we do not consider such cases in our models. Consequently, the dynamic behavior of each linear invasion could be derived based on our previous calculations, with some slight changes. The linear invasions in Section 4.1 are performed in a row-wise meander fashion, but here, the invasions are performed column-wise without any change in direction. The sizes of the energy components are derived using the equations presented in Section 4.1 by changing the row-wise parameters to column-wise parameters. Moreover, the number of invaded PEs by each linear invasion and the turn point is assumed to be equal to the size of the rectangular region columns (\(N_{ip} = N_{inv,c}\)). This implies that the invasion direction remains unchanged.

### 5. Experimental Results

The presented mathematical model prepares a strong platform in order to explore different design solutions. In order to measure the accuracy of the models, we compare the energy estimation results with results derived from a cycle-accurate simulator.
Obviously, each of these platforms offers different features by evaluating the system at different abstraction levels. The goal of our experiments is to compare different sizes of invasion controller power domains with respect to the energy consumption, invasion latency, and required hardware resources (cost). Therefore, a cycle-accurate model of the architecture is developed [Kupriyanov et al. 2007; Lari et al. 2011] and annotated with timing and power characteristics of different system components. The area and power consumption measurements are derived based on the post-synthesis results from the Synopsys EDA tool (Design Compiler), TSMC 65 nm low-power process technology and for the fast, 1.32V and 125C PVT corner. A 16 x 16 processor array is used, each PE consisting of an adder, a multiplier, a logical unit, and a branch unit. The register file of each PE consists of up to 16 general-purpose registers, and the data bit-width of the system is 32 bits. As invasion controller power domain scenarios, we chose the following: single invasion controller domains (1-iCtrl), 2 x 2 invasion controller domains (4-iCtrl), 4 x 4 invasion controller domains (16-iCtrl), 8 x 8 invasion controller domains (64-iCtrl), and a 16 x 16 invasion controller domain (256-iCtrl), in which all invasion controller are in the same power domain. Each processing unit is assigned to one power domain. Table III shows the total area overhead of the power controllers for different sizes of invasion controller power domains. It should be mentioned that by grouping invasion controllers in the same power domain, we dedicate a single power-FSM controller to all of them. But as we do not group multiple processing units together, each of them has one power-FSM controller. As the results show, by grouping more invasion controllers, we can reduce the hardware overhead, which results in less area. According to Table II, the size of each processing unit (PU) is 8,915 NAND2 equivalent gates, which is about 12 times the size of an invasion controller. This also motivates our decision to have a
single power domain per processing unit and to merge several invasion controllers into one power domain.

The next evaluation analyzes the effects of changing the size of the invasion controller power domains on the invasion latency and the overall system energy consumption. Here, the static and dynamic power consumption of the basic system components are measured using the Synopsys EDA tool as follows: \( P_{d,\text{iCtrl}} = 0.05 \text{mW}, \) \( P_{s,\text{iCtrl}} = 0.03 \text{mW}, \) \( P_{d,PU} = 2.93 \text{mW}, \) \( P_{d,PU} = 1.15 \text{mW}. \) Each of our simulation-based experiments is characterized by three different properties: (a) the number of applications, (b) the kind of applications, and (c) the total utilization ratio of the processor array (the ratio between the number of utilized PEs to the total size of array). Each experiment consists of a number of concurrent applications that are mapped to the array by performing invasions. The number of applications per experiment is randomly chosen and varies from one up to three concurrent applications. Two kinds of applications are mapped randomly: 1D applications (e.g., FIR filters) needing linear invasions and 2D applications (e.g., matrix multiplication) needing rectangular invasions. The kind of application is also decided randomly. All applications are executed on the system for the same time duration. The third parameter for each experiment is explained by the total utilization ratio, which is the percentage of the array that is used by different applications in each experiment. Different utilization ratios are 10%, 30%, 50%, 70%, and 90% of the array size. For each utilization ratio, 10,000 application scenarios were experimented upon, where each experiment is also run for the following power scenarios: without power gating, with power gating, and for different invasion controller power domain sizes.

Figure 10 shows the average invasion latency per invaded PE for different utilization ratios. This metric shows the average time necessary to invade one PE. As expected, applying power gating may increase the average invasion latency. Also, this effect is much less for larger sizes of the invasion controller power domain, especially when a large number of PEs is invaded over time. But in the case of domains larger than 16 invasion controllers, the reduction of the invasion latency diminishes. This is a crucial observation when considering the energy consumption of the invasion controllers (see Figure 11). As can be seen from the results, by grouping multiple invasion controllers in each power domain, the energy consumption is improved (compared to the single invasion controller scenario). This is due to the high timing delays imposed by the power switching of single invasion controllers, which prolong the invasion time, hence, keeping the invasion controllers in the powered-on mode for a longer time and thereby increasing the total energy consumption. In general, we can see that the energy is...
Fig. 11. Energy consumption of the invasion controllers for different sizes of invasion controller power domains.

reduced by increasing the domain size up to the 16-iCtrl version. But from this point, there is a slight increment in the energy consumption by increasing the invasion controller power domain size. The observed minimum energy consumption in each case is the result of the total amount of turned-on iCtrls in the invasion/retreat time. Although the timing overhead of invasion is reduced by increasing the size of the iCtrl power domains in Figure 10, it does not necessarily lead to less energy consumption. In Figure 11, for each array utilization ratio, the energy consumption of the iCtrls shows two different behaviors: a phase of energy consumption reduction down to a minimum point, and a phase of slight energy consumption increment. In the case of the reduction phase, the timing overhead plays a dominant role. Therefore, reducing the timing overhead by grouping more iCtrls in the same power domain results in energy consumption reduction, despite the fact that the power consumption is increased by increasing the number of turned-on elements. As observable in Figure 10, the improvement in the timing overhead diminishes when moving from the 16-iCtrl scenario to the 256-iCtrl scenario. As a result, from this point, the power consumption plays a dominant role in the energy consumption of the iCtrls. By increasing the size of the iCtrl power domain, the power consumption is increased significantly, but the timing overhead is improved slightly, which results in increasing the energy consumption. The results in Figure 11 show the trade-off between the instant power consumption of the invasion controller power domains and the overall invasion latency. It is worth mentioning that the latency of the invasion not only affects the energy consumption of the invasion controllers, but also influences the energy consumption of the processing units. As shown in Table I, the processing units are turned on during the invasion confirmation phase and turned off when receiving retreat signals. Consequently, the time period for which they are kept powered on depends on how fast we can send the invasion-related signals.

Figure 12 shows the total energy consumption of the system. Here, we can see that the proposed application-driven power gating mechanism can reduce the total energy consumption by up to 73%. This is highly visible when only a small fraction of the system is invaded (e.g., 10% or 30% of the array size). In such cases, although the dynamic energy consumption remains unchanged, there remains a great deal of static energy consumption when no power gating is used. This static energy is reduced significantly by turning off the unused components in the system. Especially, in situations with low resource utilization, a large portion of the system that is never used is still powered on and causes static energy consumption. In fact, the difference between the static energy
Fig. 12. Comparison of total energy consumption in the case of applying power gating (with different invasion controller power domain sizes) and without power gating. The maximum gained power reduction is noted, in the case of each array utilization ratio.

The energy consumption of power-gated and non-power-gated versions arises from the non-utilized portion of the system (mainly idle PUs). This also motivates our decision for applying such a fine-granular power gating approach to PU components. If we group multiple PUs into the same power domain, it may result in keeping some idle PUs in the turned-on mode. Whereas applications normally run for very long periods, this leads to a considerably higher energy consumption, due to the static energy consumption of turned-on PUs. By increasing the utilization ratio, this difference decreases, due to the fact that a higher portion of the system is powered on even in the cases of the power-gated versions.

Another interesting point in Figure 12 is the trend of lowering the total energy consumption by increasing the invasion controller power domain size. Despite the results shown in Figure 11, the energy consumption reduction continues until a single power domain remains for all invasion controllers in the array, which shows the minimum energy consumption for all utilization ratios. This is due to the effects of the invasion latency on the energy consumption of the processing units. As previously mentioned, with longer invasion latencies, the processing units stay in the powered-on mode for a longer time, which results in a higher energy consumption. As a result, although the energy consumption of the invasion controllers increases when moving from the 16-iCtrl version to the 256-iCtrl version, the decreased invasion latency (according to Figure 10) also lowers the overall energy consumption of the processing units. The energy consumption of the processing units dominates the energy consumption of the invasion controller so that we do not see the same trade-off as in Figure 11. Therefore, one of the designs from 16-iCtrl to 256-iCtrl can be chosen depending on the hardware cost and the ease of implementation. In case of the 1-iCtrl version, we can even observe a slight increase of dynamic energy consumption compared to other power-gated versions. This increase is more visible when a higher portion of the system is invaded, causing long invasion latencies, and therefore increasing the total dynamic energy consumption. This is especially crucial for the energy consumption...
of the invasion controllers, as these are active for a much longer time because of incorporating the power switching latencies in each invasion step.

Table IV compares the results of Figure 12 with the estimated energy measurements derived from the presented models in Section 4. The average number of captured PEs by linear and rectangular invasions in the simulation-based experiments is calculated for each occupation ratio, namely $N_{inv_lin}$ and $N_{inv_rect}$. Then, these two values are fed to the models to derive the energy consumption of the system in each case. As the presented models do not support concurrency in applications, we calculate the energy consumption of linear and rectangular invasions for the derived values separately. Table IV shows the absolute estimation errors for the presented models, where error is calculated as $e = \frac{|E_{sim} - E_{mod}|}{E_{sim}}$. $E_{sim}$ and $E_{mod}$ are the energy consumption values derived from the simulator and the mathematical models, respectively. Although some features, such as applications concurrency, are dropped out of the models, the estimation error is significantly low (maximum 3.6%). Overall, in most of the cases, the estimated energy from the model is less than the derived results from the simulator. It could be observed that the estimation error is reduced for lower occupation ratios and coarser iCtrl power domain sizes, and different ratios of occupation were studied.

### Table IV. Comparison of the Analytical Model for Energy Estimations and the Results Derived by Simulation

<table>
<thead>
<tr>
<th>iCtrl power domain size</th>
<th>Occupation ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10%</td>
</tr>
<tr>
<td>1-iCtrl</td>
<td>simulation</td>
</tr>
<tr>
<td></td>
<td>analytical model</td>
</tr>
<tr>
<td></td>
<td>error</td>
</tr>
<tr>
<td>4-iCtrl</td>
<td>simulation</td>
</tr>
<tr>
<td></td>
<td>analytical model</td>
</tr>
<tr>
<td></td>
<td>error</td>
</tr>
<tr>
<td>16-iCtrl</td>
<td>simulation</td>
</tr>
<tr>
<td></td>
<td>analytical model</td>
</tr>
<tr>
<td></td>
<td>error</td>
</tr>
<tr>
<td>64-iCtrl</td>
<td>simulation</td>
</tr>
<tr>
<td></td>
<td>analytical model</td>
</tr>
<tr>
<td></td>
<td>error</td>
</tr>
<tr>
<td>256-iCtrl</td>
<td>simulation</td>
</tr>
<tr>
<td></td>
<td>analytical model</td>
</tr>
<tr>
<td></td>
<td>error</td>
</tr>
</tbody>
</table>

Note: All measurements are given in mW · s. In both cases, a 16×16 TCPA, different iCtrl power domain sizes, and different ratios of occupation were studied.
error rates. Therefore, the presented models can be reported as the lower-bound energy consumption of the system.

Figure 13 concludes our results by comparing the average timing, area, and the energy consumption of each explored architecture. The vertical axis represents the area cost in terms of NAND2 equivalent gates. As it can be seen, there is a slight hardware overhead by using our adaptive power gating method. This overhead is reduced by increasing the size of the invasion controller power domains. Of course, this hardware overhead cost is compromised by reducing the energy consumption of the system, which is illustrated by the size and color of the dots in the chart. Generally, higher energy reduction is gained by using invasion controller power domains of size 16 to 256 invasion controllers. Compared to the non-power-gated version, there is a huge timing overhead when using the 1-iCtrl power domains and a slight timing overhead for the large domains, such as 64-iCtrl or 256-iCtrl. As explained before, this timing overhead plays a crucial role in increasing the power consumption of the processing units.

6. CONCLUSIONS
We have presented a low-power, many-core architecture design consisting of an array of tightly-coupled processing elements. Here, processing elements are equipped with resource-aware hardware components, called invasion controllers, that evaluate the availability of processing resources and then acquire and reserve the resources for each application, based on the required level of parallelism. We have presented an adaptive methodology on how to benefit from this paradigm inherently for power control of different components in a system. Two different kinds of power domains were realized in the system: processing unit power domains, which control the power state of the processing units, and invasion controller power domains, which control the power status of the invasion controllers. An accurate mathematical model for estimating the energy consumption of the system was presented. The estimation error of the presented models compared to the simulation results is less than 3.6%. We have shown that by placing multiple invasion controllers in the same power domain, we can make a trade-off between the timing latency of the resource exploration (which is called invasion latency), the hardware cost of the system in terms of area, and the energy consumption.
of the system. The overall energy consumption of the system showed a steady reduction while increasing the size of the invasion controller power domains. This trend is also followed in the case of the invasion latency.

In the future, we would like to extend this approach with techniques like dynamic-voltage frequency scaling and adaptive body-biasing of the PEs, by considering the computational performance demands of the applications, and with on-time monitoring of the activity inside PEs.

REFERENCES


Received March 2012; revised July 2012; accepted September 2012