Self-Adaptive Hybrid Dynamic Power Management for Many-Core Systems

Muhammad Shafique, Benjamin Vogel, Jörg Henkel
Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
{muhammad.shafique, benjamin.vogel, henkel}@kit.edu

***Abstract*—We present a self-adaptive, hybrid Dynamic Power Management (DPM) scheme for many-core systems that targets concurrently executing applications with what we call "expanding" and "shrinking" resource allocations as, for example, in [13]-[15][27]. To avoid frequent allocation and de-allocation, it enables applications to temporarily reserve their resources and to perform local power management decisions. The expand-to-shrink time periods and resource demands are predicted on-the-fly based on the application-specific knowledge and the monitored system information. Experimental results demonstrate up to 15%-40% Energy-Delay[2] Product reduction of our scheme compared to state-of-the-art power management schemes like [4][8]. Self-adaptive local power-management decisions make our scheme scalable for large-scaled many-core systems as illustrated by numerous experiments.**

I. Introduction and Motivation

Processor architecture is rapidly moving towards many-core systems that will feature 100s–1000s of cores connected via an on-chip network [1][2][10]. Following the commercial trends (Tilera chip with 100 cores [5], Nvidia GPUs with 1024 cores [6]), it is predicted by the ITRS that the next generation many-core systems will feature 1460 cores in 2020 to 5900 cores in 2026 [1]. An increasing numbers of cores lead to serious power related issues. System designers and architects address the power problem using several low-power techniques like (1) power-gating[1] considering multiple sleep states, providing leakage savings vs. wakeup overhead tradeoff; (2) clock gating; (3) dynamic voltage and frequency scaling (DVFS) with multiple V/F levels; and (4) body biasing [20]. However, effective power management in many-core systems has become an intricate problem, especially in large-scale computing systems. In the following, we discuss two key power management challenges in emerging many-core systems: (1) scalability and complexity issues to determine appropriate power state of the cores; (2) power/energy efficiency issues related to applications with suddenly changing workloads.

A. Emerging Power Management Issues and Challenges

(1) Scalability Issue in Power Management of Many-Core Systems: A large number of cores (100s–1000s) leads to an explosion of power management decision space[2] that limits the applicability of state-of-the-art centralized power management policies [8][9]. Moreover, DVFS scaling potential is diminishing due to the shrinking gap between nominal and threshold voltages and high overhead of voltage regulators in densely integrated chips with 100s of cores [8]. For example, a 100 core system with each core exhibiting 3 power states (on, off, state-retentive) results in theoretically 3^{100} possible power management options. While fast search and optimization heuristics may be applied to prune the decision space, recent trends in power-management for many-core systems have shown a paradigm shift towards scalable and distributed power management techniques [10][11]. Moreover, in order to cope with the rapidly growing complexity of future computing problems, IBM's autonomic computing initiative builds the consent towards self-optimizing/self-adaptive systems, such that different system components (e.g., cores) adapt and optimize themselves autonomously to operate efficiently as a whole [7][25][27]. *Therefore, power management in emerging many-core systems needs to be distributed & self-adaptive such that power states of the cores can be autonomously / efficiently controlled by their owner applications*. To efficiently use the plethora of computing resources in many-core systems, these owner applications typically exhibit self-optimizing resource allocation that happens to be called *"expanding and shrinking applications"* in our case. Such a trend of hosting multiple simultaneously executing expanding/shrinking applications on advanced many-core systems has already been discussed by the research community [13]-[15][27].

In the following, we briefly introduce *expanding* and *shrinking* applications and power management issues related to them.

(2) Power Management Issues related to the Expanding and Shrinking Applications: applications ***expand*** (i.e. demand more resources from the *local* resource manager) and ***shrink*** (i.e. return dispensable resources to the *local* resource manager) at run time w.r.t. their resource requirements due to a varying degree of required parallelism as a result of abruptly changing workloads and performance/power constraints[3]. Fig. 1 shows an abstract scenario where two applications executing on a many-core system are expanding and shrinking at run time. Overlapping or consecutive expand periods correspond to *resource competition* among simultaneously executing applications (i.e. **A** and **B** in Fig. 1). Therefore, application **B** may shrink in low workload conditions to facilitate application **A**; see Scenario (3).

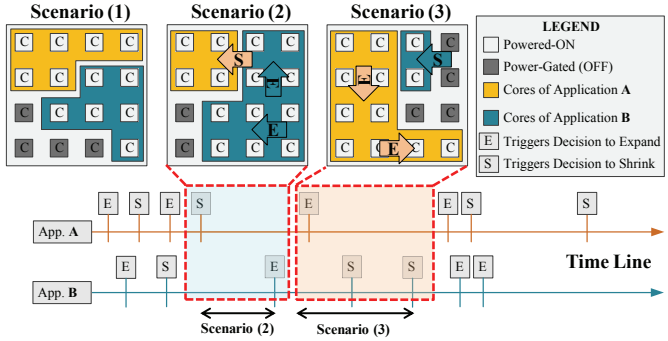

Fig. 1 Expanding and shrinking applications on a many-core system: different scenarios show varying expand-to-shrink and shrink-to-expand time periods

The *expand* and *shrink* are triggered by the application using special commands that are part of the application's code using special programming language extensions, like in resource-aware programming paradigms (e.g. [24]) where applications specify their resource requirements in the program. The concept is based on leveraging the application-specific knowledge to derive hints towards resource requirements. An application triggers *expand* when its throughput constraints are not fulfilled and therefore requires extra cores. A *shrink* is triggered by the application when its computational requirements are fulfilled and it no longer requires all cores. This way, each application attempts to adapt its own resource requirement. After receiving the *expand* and *shrink* commands containing the information about the demanded or dispensable cores, an *application's local resource manager* (in a distributed resource management paradigm) performs resource allocation/de-allocation decisions through inter-application negotiation. These inter-application negotiations involve

[1] Shutting down the idle core or parts by switching off their power-supply using sleep transistors; power-gating incurs a wakeup latency and energy overhead that can be significant depending upon the gated region.

[2] Depends upon the number of power states, number of cores, and recorded statistics like power, performance, workload history, etc.

[3] Examples: scalable and slice-parallelizable video encoders, scalable tracking algorithms, database applications with search and sorting functions for a large-size data set, etc.

978-3-9815370-0-0/DATE13/©2013 EDAA

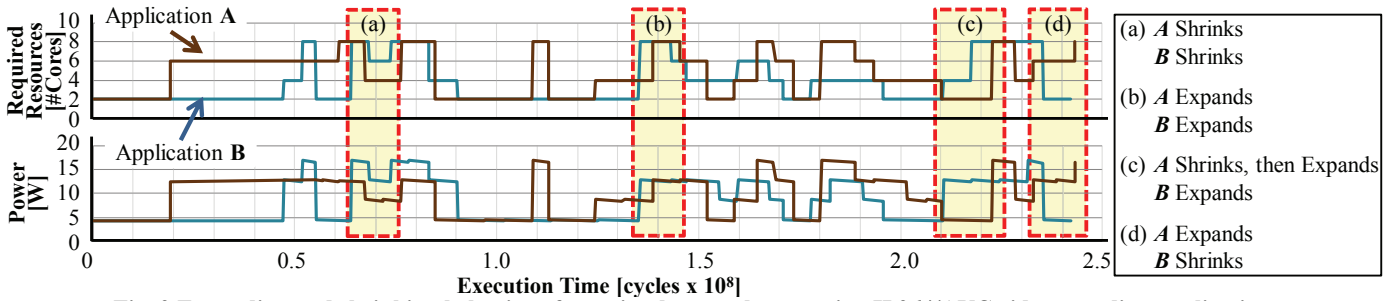

Fig. 2 Expanding and shrinking behavior of two simultaneously executing H.264/AVC video encoding applications (see experimental setup in Section IV)

interaction between the local resource managers of different simultaneously executing applications based on their requirements [13][23]. To efficiently utilize the allocated cores, these expanding/shrinking applications are designed (e.g., using Intel's threading building blocks [12]) to dynamically adapt their degree of parallelism to the number of allocated cores at run time [13][15].

Energy-Related Issues: Simultaneous execution of such expanding and shrinking applications leads to highly unpredictable resource demands/utilizations and *frequent shrink-to-expand and expand-to-shrink time periods* (i.e. time between an expand operation and an immediately following shrink operation). This ultimately prohibits state-of-the-art distributed power management schemes [10][11] to achieve effective power management due to (i) significant wakeup overhead; (ii) lack of prediction accuracy/effectiveness; (iii) decreased potential of chip-level power gating [3]. We illustrate the power-related issues associated with the expanding/shrinking applications with the help of our experimental case study in Fig. 2.

Our experiments in Fig. 2 illustrate multiple scenarios of expand and shrink by means of two parallelized H.264 video encoder applications (as **A** and **B**) from the *parsec* benchmark. The varying core requirements denote the varying throughput constraints of two applications and corresponding expand and shrink operations. Let us assume the case where application **A** is shrinking (Fig. 2 (c)), it may power-gate its dispensable cores or may return them to its *local* resource manager. Since this application did not have the expand/shrink knowledge of the simultaneously executing applications that are competing for the resources (cores)[4], this may lead to power/energy inefficiency as discussed in the following cases:

Case-1: Application A power-gated its dispensable cores: now if application **B** expands and immediately requires the resources, it leads to frequent ON-OFF switching of the dispensable cores of application **A** that results in power/energy inefficiency; see Fig. 2 (c). A similar case may happen if application **A** expands again; see Fig. 2 (d).

Case-2: Application A returns back its dispensable cores to its local resource manager that performs de-allocation: now if application **B** acquires these dispensed cores as a result of an *expand* and right afterwards, application **A** expands again, application **A** does not receive back its cores and it suffers from low energy efficiency and severe performance degradation; see Fig. 2 (c) and (d).

In the above-discussed scenarios, it might be energy-wise beneficial for application **A** (which is shrinking in scenario Fig. 2 (c)) to **temporarily reserve** its cores instead of releasing them. However, this requires:

1) A *scalable per-application resource reservation policy* that accounts for the knowledge of other simultaneously executing applications' expand/shrink time periods; and
2) Accurately predicting the expand-to-shrink and shrink-to-expand time periods to avoid unnecessary resource blockage such that other applications do not suffer.

[4] Fig. 2 (b) illustrates a resource competition scenario.

Another power-related issue of *frequent* expand/shrink operations is that they lead to an increased frequency of local resource management decisions, which results in excessive latency and energy overhead due to:

a). Frequent resource negotiations among local resource managers of simultaneously executing expanding and shrinking applications, decision logic, search for free cores, continuous adaptation during the application execution, etc.;
b). Resource allocation and de-allocation: starting a new thread of the expanding application on the newly-allocated core and stopping a thread of the shrinking application on this core, task migration, etc.

The latency overhead of applications' local resource management ranges from 0.1ms to 10ms at 2GHz with an energy cost of 20 µJ per activation of the local resource management (assuming the technique of [14]). Consequently, a long latency also results in degraded energy efficiency of the application. Therefore, if expand-to-shrink and shrink-to-expand time periods of different applications are correctly predicted and jointly considered, significant energy savings can be obtained by avoiding frequent local resource management decisions and sudden variations in the power states of dispensed cores.

B. Summary of Research Challenges

Summarizing the above analysis and discussion, **the key research challenges** for efficient power management of many-core systems hosting simultaneously executing expanding and shrinking applications are:

a) Enabling a scalable per-application resource reservation policy while accounting for the expand/shrink time periods of other simultaneously executing applications;
b) Accurately predicting the expand-to-shrink and shrink-to-expand time periods, while leveraging the application-specific knowledge;
c) Selection of appropriate sleep states for different cores in cases of highly *un-aligned* and *frequent* expand-to-shrink time periods.

The goal is to maximize the energy efficiency of resource-competing applications in a self-adaptive way through temporary resource reservations. Here, the energy efficiency is defined as the *Energy-Delay^2 Product* (ED^{2}P).

C. Overview of Our Novel Concept and Contributions

This paper combats with the above-discussed power issues and challenges in many-core systems with our novel **SEAD (Self-Adaptive Hybrid Dynamic Power Management) Scheme** that employs a per-application dynamic power-management policy in conjunction with continuous system feedback from other simultaneously executing applications, thus hybrid in nature. SEAD allows different owner applications to autonomously control the power states of their cores, while jointly accounting for the expand/shrink properties of the owner and other simultaneously executing applications. SEAD incorporates:

1. **Virtual Power Gating (ViPG) Manager** that temporarily reserves the cores of its shrinking owner application instead of releasing/unblocking them. We call this process of reserving *virtual power-gating* of cores, i.e. these cores are *unavailable* from the local resource manager's perspective and are *not* returned during the shrink operation, while at the same time *available* from the owner application's perspective for its execution. For (power-

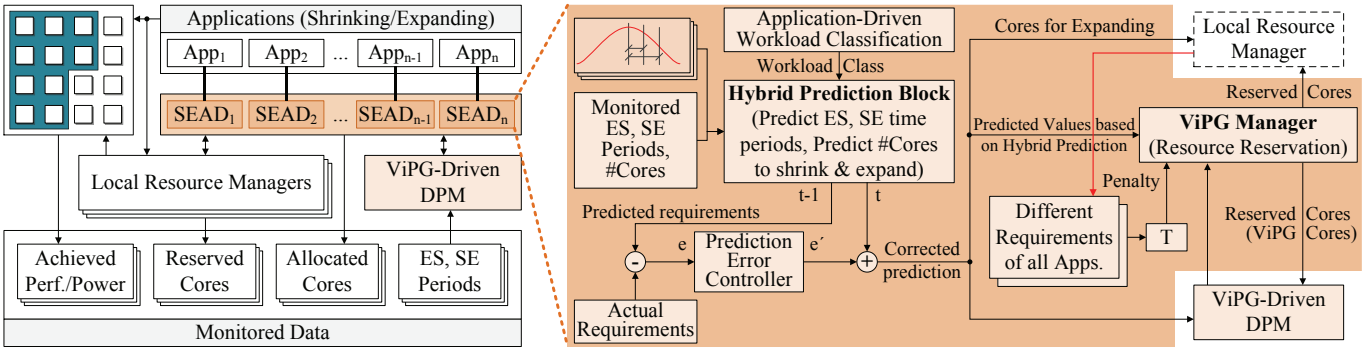

Fig. 4 Operational flow of our SEAD scheme (novel contribution is highlighted in colored boxes)

wise) efficient ViPG decision, an application's ViPG manager requires prediction of the next expand operation (number of cores, expand time period) of the shrinking owner application. To facilitate this, our SEAD scheme employs:

2. **A Hybrid Prediction Technique** that *jointly* accounts for the *application-level knowledge* of expand/shrink behavior (number of cores and time periods, workload characterization, etc.) and *run-time monitored hardware-level knowledge* (expand, shrink, execution time statistics) in order to provide an improved prediction of expand and shrink behavior under uncertainties (like sudden application workload fluctuations). *A combination of both application-level and hardware-level information defines the hybrid nature of the prediction*.
3. **ViPG-Guided Power-Gating** that may decide to power-gate cores from a virtually power-gated state depending upon the time period between shrink and expand of an application. For this, the information available at the ViPG-manager (like prediction of expand/shrink time periods and resources) can be exploited to obtain a robust prediction for the power-gating of cores.

The SEAD scheme minimizes the overall energy considering the dynamic power of expanding and shrinking applications, power consumption during resource allocation/de-allocation, leakage power, wakeup overhead, etc. under run-time scenarios of frequently expanding/shrinking applications. It is evaluated for energy-efficiency and performance compared to state-of-the-art using ED^{2}P, scalability w.r.t. number of cores and number of expanding-shrinking applications, and accuracy of hybrid prediction (see Section V).

Before proceeding to the details of our novel SEAD scheme, we will present our system models for clarity of the discussion.

II. System Models

Architecture and Power-Gating Model: A homogeneous many-core system consisting of N_C total number of cores. The cores are connected via an on-chip network. Each core can be power gated using a multiple-sleep state power-gating circuitry. The power-gate model is based on a model with five power states: P = {S_{OFF}, S_{IR}, S_R, S_I, S_{ON}}, i.e. OFF, Irretentive, Retentive, Idle, and ON states (see Fig. 3).

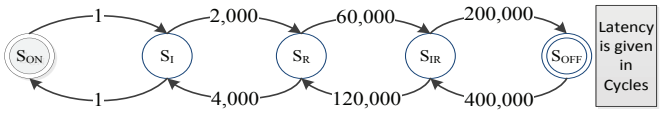

Fig. 3 Used power state machine with latencies of state transition

Multiple sleep states (achieved by back body biasing) provide a tradeoff between leakage savings and wakeup overhead. S_I performs only clock gating. In S_R, the PLL is active and the L1 cache is flushed. In S_{IR}, PLL is switched off and L2 cache is also flushed. In S_{OFF}, the state of the core is saved on a small on-chip SRAM powered by the I/O supply to allow faster wakeup times during the power up phase. A similar model can also be found in modern Intel processors [17].

Application Model: The many-core system hosts a set of simultaneously executing applications $A=\{a_1, a_2, ..., a_n\}$ with expanding and shrinking capabilities. $N_{C,i}$ denotes the number of cores owned by the application a_i. The time period and core requirements between expand and shrink are denoted as T_{ES} and N_E. The time periods and core requirements between shrink and expand are denoted as T_{SE} and N_S, respectively. Each application may temporarily reserve N_{Res} number of cores through ViPG. Each application a_i has a set of workload classes C_i, obtained using offline statistical analysis of execution properties w.r.t. the input data characterization.

III. Our SEAD Scheme

Fig. 4 shows an overview of the operational flow of our SEAD scheme (novel contribution in colored boxes) which raises the abstraction level of power management to the application level in a hybrid fashion, i.e. scalable using per-application power managers with system feedback about other simultaneously executing applications. The owner applications can independently control the power states of their cores. SEAD allows applications to locally manage the power states of their cores in a two-step fashion: (1) performing virtual power-gating (i.e. resource reservation) using the ViPG manager; and (2) power-gating in an appropriate sleep state. For this, SEAD requires the prediction of expand-to-shrink and shrink-to-expand operations.

Our scheme works via four major phases (see details in later sections):

1) Predicting the time period and number of required cores for expand-to-shrink and shrink-to-expand operations in a hybrid way, i.e. by jointly accounting the application-level and hardware-level knowledge;
2) Performing the ViPG operation on potentially dispensable cores by computing their reservation benefit, such that these cores are not available to the local resource manager for shrinking;
3) Accounting for the system feedback as a penalty in the ViPG benefit function to account for the performance/energy degradation of other simultaneously executing applications;
4) Moving the power states of cores from virtually-power-gated to power-gated in an appropriate sleep state, depending upon the predicted shrink-to-expand time period.

D. Hybrid Prediction of Expand-to-Shrink and Shrink-to-Expand Operations

Once an application decides to shrink, SEAD predicts the upcoming shrink-to-expand (SE) and expand-to-shrink (ES) time periods and numbers of cores required in each time period. History-based prediction techniques may lead to significant miss-predictions in case of suddenly changing workloads [22]. Application-based prediction techniques (like [22][26]) may handle such workload fluctuations, but do not account for the architectural information (like monitored statistics, idle period fluctuations, etc.), thus may lead to miss-predictions and consequently less-effective power management in many-core systems. To overcome the limitations of both application- and history-based predictions, SEAD employs a hybrid prediction technique. Fig. 5 shows the pseudo-code of our hybrid prediction technique. The inputs are the application-dependent workload classes C, probability-density functions (PDF) of

different parameters like (T_{SE}, T_{ES}, N_{Res}, N_E and N_S) associated with expand-to-shrink and shrink-to-expand operations, and the monitored history data H of these parameters. The output is the set of predicted values of these parameters using the hybrid prediction technique. First the workload class is matched based on the current data set at the shrink/expand point and an appropriate PDF is selected based on the workload class (i.e. low, mid, or high, as discussed above); line 3. These workload classes depend upon the application type and are obtained using an offline statistical analysis of the application workload. Note: a change from low to high workload class denotes an abrupt change in the workload and the core requirements of an application vary significantly when moving from one workload class to another.

1. **HybridPrediction** (*Input*: application-based probability distribution functions (PDF) and monitored history (H) of ES and SE operations for a given application, i.e. N_S, N_E, T_{SE}, T_{ES}, N_{Res}; workload classes C; *Output:* predicted values for ES and SE operations)
2. $\forall v \in V = \{T_{SE}, T_{ES}, N_{Res}, N_S, N_E\}$ {
3. $c \leftarrow$ matchWorkloadClass(C, v.PDF);
4. $v.P_A \leftarrow F(c, \mu+2\sigma)$;
5. $v.P_H \leftarrow avg(v.H)$;
6. $v.P \leftarrow (\alpha \times (v.p_A)) + ((1-\alpha) \times (v.p_H))$;
7. $v.P \leftarrow$ applyCorrection(v.p,v.p_{t-1}); }
8. return $V = \{T_{SE}, T_{ES}, N_{Res}, N_S, N_E\}$;

Fig. 5 Pseudo-code of the hybrid prediction technique

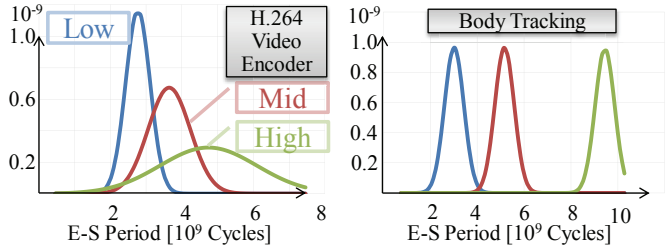

Fig. 6 Probability density functions (PDFs) of expand-to-shrink time periods for different workload classes of two test applications

Afterwards, the application-based prediction of different parameters (like T_{SE}, T_{ES}, N_{Res}, N_E and N_S) is computed as a highly-probable value obtained using the probability distribution function (PDF, see Fig. 6) with a probability of 0.94 (assuming a Gaussian distribution) using *"$\mu+2\times\sigma$"*; line 4. μ denotes the mean of distribution and σ denotes the standard deviation. The history-based prediction is obtained as the average of the history window; line 5. Given a user-defined weighting parameter *'α'*, the hybrid prediction is obtained as a weighted sum of application-based and history-based predictions; line 6. Since the prediction may deviate from the actual requirements, our scheme employs a prediction correction using a simple PID-based prediction error controller (see Fig. 4). The controller parameters (K_P, K_I, K_D) can be computed using the Ziegler-Nichols Method [21] with the settings of Eq. 1.

$$K_P = 0.6 \times K_{PC};\ K_I = K_P / (0.5 \times T_C);\ K_D = K_P \times (0.125 \times T_C);$$
$$K_{PC} = 0.8;\ T_C = 2; \quad (1)$$

The prediction error is then added in the new prediction as a correction factor (line 7). Note that this controller loop only works on the prediction error, which adapts the quality of prediction. Therefore, this is not dependent upon the periodic or non-periodic natures of the applications' expand and shrink operations. The set of parameters is forwarded to the ViPG manager and ViPG-driven DPM (see Fig. 7).

E. Virtual Power-Gating (ViPG) of Dispensable Resources

Depending upon the shrink amount (i.e. number of dispensable cores), the predicted SE time period (T_{SE}), and the upcoming expand requirements in terms of cores (N_E), the ViPG manager computes the reservation benefit of the dispensable cores. SEAD employs ED^{2}P (Energy-Delay^2 Product) as the energy function in the profit computation since reservation affects both the energy and performance of applications. In this way, if an application is going to require its dispensable cores soon, these cores will be immediately available to it without incurring additional overhead of resource allocation/de-allocation, delayed or no additional speedup depending upon resource manager decisions and demands of other simultaneously executing applications. ViPG can potentially affect the performance of another expanding application that may result in high energy consumption or alternatively the resource manager spends more power in searching for cores for the expanding application. Therefore, ViPG computes the **reservation benefit** which consists of four parameters:

1) *Energy savings due to reservation of N_{Res} cores of application a (Ef_{Res})*: since the application will reserve the cores for its predicted upcoming expand operation; this application will start with improved energy efficiency due to readily available cores.
2) *Energy savings due to reduced effort of the local resource manager (Ef_{RM})*: ViPG cores lead to reduced decision space of the local resource manager, thus providing energy savings.
3) *Energy savings due to reduced effort of shrink and expand operations (Ef_{SE})*: due to the reservation, the energy overhead of de-allocation in the shrink operation and re-allocation in the expand operation are saved.
4) *Energy penalty due to reduced energy efficiency of other simultaneously executing applications due to reduced number of available cores to the resource manager ($Ef_{Penalty}$)*: core reservation by one application may hurt the energy efficiency of other simultaneous ly-executing applications. Therefore, SEAD employs a penalty cost of the application with potentially worst loss of energy efficiency in the reservation profit.

Formal Problem: Given an application a_i from a set of applications $A=\{a_1, \ldots, a_n\}$, the objective is to select a set of cores (N_{Res}) from the total allocated cores (N_C) of the owner application a_i that can be put into ViPG mode at the shrink operation such that, the ED^2P of a_i is minimized for the upcoming expand operation. To further improve the energy efficiency, the objective is to determine an appropriate sleep state $p \in PS = \{S_{OFF}, S_{IR}, S_R, S_I\}$ for each of the reserved core.

Algorithm: Fig. 7 shows the pseudo-code of our SEAD scheme. SEAD instance is associated with an application a, which receives N_C cores from the resource manager. The output of SEAD is the number of reserved cores and their corresponding power states. First, the hybrid prediction is performed for different parameters like T_{SE}, T_{ES}, N_{Res}, N_E and N_S, which are associated with expand-to-shrink and shrink-to-expand operations (line 2). These parameters are used for computing the reservation energy profit and sleep state (lines 6 and 22). The ViPG loop (lines 4-19) iterates over all the available cores in terms of core steps *CS*, i.e. number of cores required to take the application a to the next speedup step (this highly depends upon an application's parallelization potential and data flow properties). For each *cStep* from the set of core steps *CS*, the energy benefit of the reservation is computed as the reservation profit (*resProfit*, line 6-15) and the *cStep* number of cores to be reserved (i.e. put in ViPG state) if the reservation profit is positive (line 16). Since ViPG makes local decision for the owner applications, it may not take system-wide optimal decisions. Therefore, additional information (i.e. resource deficiency and loss in energy efficiency of other simultaneously executing applications) is forwarded from the resource manager to the ViPG manager. The ViPG manger incorporates this energy loss information in the reservation profit function as a **penalty term** (see lines 7-14). This makes SEAD a hybrid and self-adaptive power-management scheme. Depending upon the predicted time period between shrink and upcoming expand of an application, **ViPG-driven DPM** may decide to power-gate the virtually-power-gated cores depending upon the amortization of the wakeup overhead. The key is to leverage the ViPG-level information (like T_{SE} and N_{Res}) to obtain an

improved prediction of sleep periods of ViPG cores. This will avoid frequent power-on/-off switching of ViPG cores under frequently expanding/shrinking operations and highly un-aligned shrink-to-expand and expand-to-shrink time periods. Therefore, after determining the ViPG cores, SEAD determines an appropriate sleep state for them using the ViPG-driven DPM technique (lines 20-25) considering a multiple sleep state model (see Section II).

```
1.  SEAD (Input: application a, allocated resources N_C; Output:
      Reserved resources N_Res and their power states P_Res)
2.  V={T_SE,T_ES,N_Res,N_S,N_E} ← HyPred(a, a.PDF, a.H, a.C);
3.  N_Res ← 0; N_S ← 0; // initialize reservation and shrink candidates
4.  while((CS = a.getCoreSteps( ))!=∅)   {
5.    ∀cStep ∈ CS   {
      // Compute the energy benefit for reservation, resource allocation, and
      shrink-to-expand operation using the given energy function (i.e., ED²P)
6.    {Ef_Res,Ef_RM,Ef_SE} ← computeEnergyFunction(a,V);
      // Compute the energy penalty from the worst loss among all other
      applications demanding the same cStep resource
7.    Ef_Penalty ← 0;
8.    ∀z ∈ A \ a   {
9.      if(z.N_E>z.N_C)   {
10.       D ← z.N_E − cStep;
11.       Ef_Loss ← z.getEnergyLoss(D,z.N_E);
12.       if(Ef_Loss>Ef_Penalty) Ef_Penalty ← Ef_Loss; A_affected ← z;
13.     }
14.   }
      // Compute the reservation profit and perform ViPG
15.   resProfit ← Ef_Res + Ef_RM + Ef_SE − Ef_Penalty;
16.   if(resProfit>0) N_Res ← N_Res+cStep; CS ← CS \ cStep;
17.   N_S ← N_C − N_Res;
18.   }
19. }
      // Perform ViPG-Driven DPM
20. ∀c ∈ N_Res   {
21.   ∀p ∈ PS={ S_OFF,S_IR,S_R,S_I}   {
22.     if(E_benefit(T_SE,p.P_Leak)>E_overhead(p))   {
23.       PowerGate(c,p); break;   }
24.   }
25. }
```

Fig. 7 Pseudo-code of our SEAD Scheme

IV. Experimental Setup

We simulated applications from the *parsec* benchmark suite [16] that exhibit varying degree of parallelism, which is important to realize expanding/shrinking applications (see Fig. 8). We extended the *parsec* applications to model a resource-aware behavior, i.e. perform shrinking and expanding. The selected applications demonstrate considerable variation in their resource requirements and execution times depending on the input data. *bodytrack* tracks a person in a sequence of 3D-camera images. We used input sets A and B with 1000, 2000, and 4000 particles. *x264* is a parallel H.264/AVC video encoder. We combined several HD video sequences: **Mix 1** refers to *bluesky*, *station*, *eledream*, **Mix 2** refers to *in_to_tree*, *old_town_cross*, *eledream*, and **Mix 3** refers to *sunflower*, *station2*, *crowd_run*.

We used a cycle-accurate performance simulator (Gem5 [18]) with an ITRS-based power simulator (McPAT [19]) as depicted in Fig. 8. Every experiment is executed for $20x10^9$ cycles.

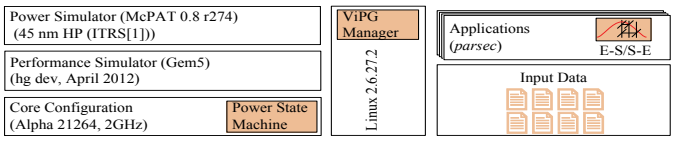

Fig. 8 Experimental Setup

V. Results and Discussion

A. Comparison to State-of-the-Art

We compare our SEAD scheme with advanced state-of-the-art power management policies [4][8] (Fig. 9 b, c, d) and the baseline system (i.e. without SEAD scheme, Fig. 9 a) for varying number of simultaneously executing applications for a given number of cores. Fig. 9 illustrates the normalized ED²P reduction of our SEAD scheme compared to the comparison partners; bars denote the average, while the line denotes the standard deviation.

Fairness of Comparison: for fairness of comparison, we provide the same power state machine to all schemes. We also allow all schemes to access the expand/shrink knowledge of all owner applications to provide them the knowledge of resource requirements.

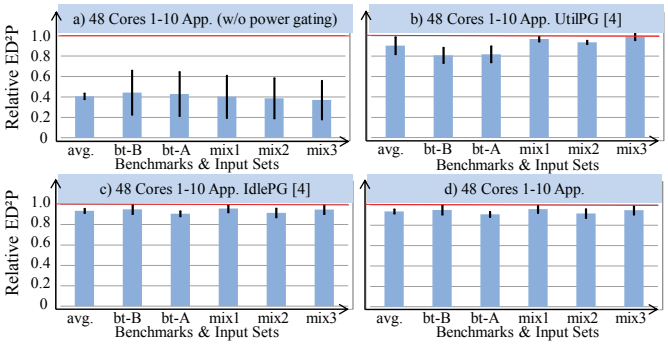

Fig. 9 Average ED²P improvements compared to state-of-the-art (bars denote average savings, line-ends show max/min savings)

1) **Comparing to the Baseline System, i.e. w/o SEAD (Fig. 9 a):** In the baseline system the applications expand and shrink, but the unused cores are active all the time. Compared to the baseline case, our SEAD scheme provides up to 96.9% (average 59.68%) ED²P reduction; see Fig. 9 a. For a more realistic evaluation, we adopt advanced state-of-the-art power management schemes UtilPG [4], IdlePG [4], and MaxBIPS [8] for comparison.

2) **Comparing to the UtilPG Scheme [4] (Fig. 9 b):** UtilPG [4] does not support shrink, i.e. application do no return their dispensable cores, rather power-gates them in case the utilization is under a predefined threshold. These power-gated cores are unavailable to other applications, and only the owner applications can activate them for usage in case of workload increase. Therefore, UtilPG suffers from energy inefficiency in case of multiple competing applications due to the unavailability of the expand/shrink knowledge of other applications. In contrast, ViPG manager in our SEAD scheme accounts for other application in the reservation benefit function thus allows shrinking in case other applications are in urgent need of the resource. As a result, compared to the UtilPG [4], our SEAD scheme provides up to 42.56% (average 10.26%) ED²P reduction; see Fig. 9 b.

3) **Comparing to the IdlePG Scheme [4] (Fig. 9 c):** IdlePG [4] is a timeout based power management scheme. The cores are power-gated if the cores' idle time exceeds a predefined threshold. Like UtilPG, IdlePG also does not account for other simultaneously executing applications. Therefore, IdlePG suffers from energy inefficiency along with significant performance degradation; especially in case of sudden expand time periods. In contrast, our SEAD leverages the application specific knowledge and monitored statistics to enhance the prediction of time period and core requirements during the expand/shrink periods. Moreover, hybrid prediction of SEAD allows for accurately predicting the overlapping expand and shrink time periods of different applications. SEAD thereby achieves up to 17.33% (average 6.97%) ED²P reduction compared to the IdlePG scheme; see Fig. 9 c.

4) **Comparing to the MaxBIPS Scheme [8] (Fig. 9 b):** MaxBIPS is a throughput optimizing power management scheme, where execution phases are predicted based on the application's prior knowledge and the workload demand of the future is unknown. The core benefit of SEAD arises from its reservation policy, as MaxBIPS will return its resources during a shrink operation. Moreover, the hybrid prediction provides an edge to our SEAD scheme by providing an accurate estimate of the upcoming expand/shrink operations. As result SEAD achieves up to 15.95% (average 6.6% %) ED²P reduction compared to

the MaxBIPS scheme; see Fig. 9 d. Note: even 7%-16% ED²P savings are significant as the comparison is purely based on the power management policy and all other system conditions are set to be same for all the comparison partners.

B. *Results for Scalability w.r.t. number of Cores and Applications*

Fig. 10 illustrates SEAD's ED²P savings normalized to MaxBIPS [8] for an increasing amount of cores (see Fig. 10 a-d) and number of simultaneously executing applications. It is worthy to note in Fig. 10a) that the ED²P savings becomes constant after 52 cores. It is due to the fact that core requirements of all applications were fulfilled. Therefore, to analyze the scalability to a large number of cores, the number of simultaneously executing applications is increased; see 22 → 33 → 44 applications when moving from 72 to 192 cores.

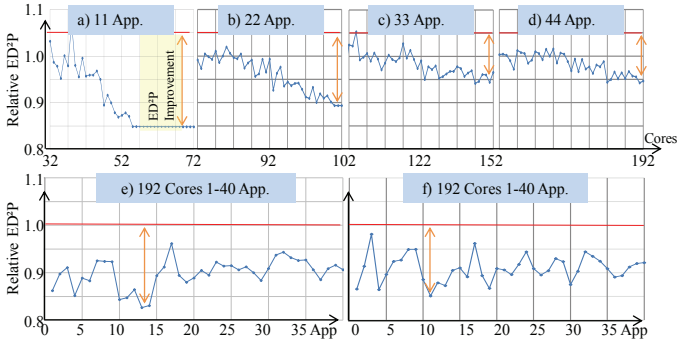

Fig. 10 : Scalability of SEAD's ED²P savings compared to MaxBIPS [8] w.r.t. number of cores and simultaneously executing applications

Fig. 10 a-d shows that SEAD achieves improved energy efficiency in most of the cases. However, in a few cases SEAD suffers from energy inefficiency. This is because of the expansion of another application by a significant amount, while another application already reserved many cores. This sudden expansion was not sufficient for the penalty function to avoid reservation. Therefore, though one application exhibits better energy efficiency, another application suffered. This resulted in overall energy efficiency by less than 5% (see Fig. 10 c, near 102 cores case). However, in case of longer executions, due to continuous system feedback, SEAD moves towards overall energy efficient operating points and the overall savings improve. In summary, Fig. 10 shows that SEAD outperforms the state-of-the-art MaxBIPS[8] scheme in almost all the cases. In Fig. 10 e) we compare SEAD against the IdlePG [4] and in Fig. 10 f) against MaxBIPS [8] for a varying amount of applications. Note, there are certain set of spikes in the ED²P savings. This is because of the fact that an application ED²P is improved in steps, i.e. to move to the next ED²P point, an application requires more than 1 core. This short-time energy penalty would also be compensated in long execution cases.

C. *Results for Prediction Accuracy of Hybrid Prediction Technique in SEAD*

The prediction accuracy is presented in Fig. 11 for SEAD and a sliding average history based predictor compared to the actual core requirements of applications. In average, SEAD accurately predicts the core requirements (avg 0.00 cores, stdev 0.57 cores). However, the sliding average predictor underestimates the core requirements by 0.94 cores, stdev 5.39 cores. The improved prediction accuracy is primarily due to the joint consideration of application-specific knowledge and hardware-level monitored statistics.

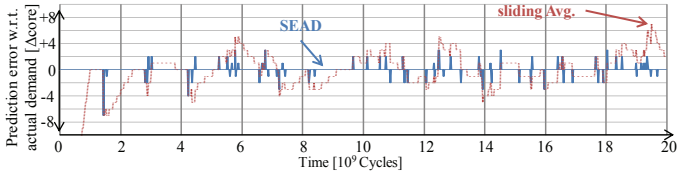

Fig. 11 : Prediction Accuracy

VI. Conclusions

We presented a self-adaptive dynamic power management scheme for scenarios of simultaneously executing applications with sudden expanding/shrinking resource requirements. It employs the novel concept of Virtual Power Gating to temporarily reserve cores along through a hybrid prediction technique to provide high energy efficiency. Compared to advanced state-of-the-art techniques [4][8], we achieve 15%-40% ED²P reduction. Our self-adaptive scheme enables applications to autonomously control the power states of their resource. This provides a foundation for scalable power management of future many-core systems.

Acknowledgment

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89); http://invasic.de.

References

[1] ITRS-International Technology Roadmap for Semiconductors, "System drivers", Available at http://www.itrs.net, 2011.

[2] S. Borkar, "Thousand core chips: a technology perspective", IEEE DAC, pp. 746–749, 2007.

[3] D. Meisner, B. T. Gold, T. F. Wenisch, "The PowerNap server architecture", ACM TOCS, 2011.

[4] N. Madan et al.: "A Case for Guarded Power Gating for Multi-Core Processors", HCPA2011.

[5] Tilera Corporation [www.tilera.com], "Tile-GX Processor Family", 2011.

[6] Nvidia Tesla [www.nvidia.com]: "Tesla Processor Family", 2011.

[7] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology", Computing Systems, pp. 1–40, 2001.

[8] C. Isci et al., "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget", MICRO, 2006.

[9] K. Meng et al., "Multi-optimization power management for chip multiprocessors", IEEE PACT, 2008.

[10] K. Ma et al., "Scalable Power Control for Many-Core Architectures Running Multi-threaded Applications", IEEE ISCA, 2011.

[11] J. Sartori, R. Kumar, "Distributed Peak Power Management for Many-core Architectures", IEEE DATE, 2009.

[12] J. Reinders, "Intel threading building blocks", O'Reilly & Associates, 2007.

[13] S. Kobbe et al., "DistRM: Distributed Resource Management for On-Chip Many-Core Systems", CODES+ISSS, pp. 119-128, 2011.

[14] H. Shojaei et al., "A Parameterized Compositional Multi-dimensional Multiple-choice Knapsack Heuristic for CMP Run-time Management", DAC, 2009.

[15] G. Sabin et al., "Moldable parallel job scheduling using job efficiency: An iterative approach", JSSPP, vol. 4376, pp. 94-114, 2007.

[16] C. Bienia, "Benchmarking Modern Multiprocessors", PhD thesis, Princeton University, 2011

[17] George et al.: "PENRYN: 45-nm next generation Intel Core 2 Processor," in Proc. ASSCC, Nov. 2007.

[18] Binkert et al., "The gem5 simulator", ACM SIGARCH Computer Architecture News, volume 39, no. 2, 2011.

[19] Li et al., "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", MICRO, 2009.

[20] H. Singh et al., "Enhanced leakage reduction techniques using intermediate strength power gating", IEEE Transaction on VLSI, 2007.

[21] M. A. Johnson. M. H. Moradi, "PID Control: New Identification and Design Methods", Springer, 2005.

[22] M. Shafique et al., "Adaptive Power Management of On-Chip Video Memory for Multiview Video Coding", DAC, pp. 866-875, 2012.

[23] B. Yang, et al., "Multi-application multistep mapping method for many-core network-on-chips", NORCHIP, pp. 1-6, 2010.

[24] L. Mureau et al., "Resource aware programming", TOPLAS'05.

[25] F. Sironi et al, "Metronome: Operating system level performance management via self-adaptive computing", DAC, 2012.

[26] X. Liu et al, "Chameleon: application-level power management," TMC, 2008.

[27] J. Henkel et al., "Invasive Manycore Architectures", ASP-DAC, 2012.