Voltage Island-Aware Energy-Efficient Scheduling of Parallel Streaming Tasks on Many-Core Processors

Nicolas Melot,
Linköping University, Sweden
⟨name.surname⟩@liu.se

Christoph Kessler
Linköping University, Sweden
⟨name.surname⟩@liu.se

ABSTRACT

The quadratic drop of power consumption associated with the decrease of voltage supply provides a popular way to optimize applications for energy-efficiency by decreasing cores' frequency and voltage. However, this requires additional hardware within the chip so that to regulate voltage and frequency for each part of a circuit that can be scaled. Because of the expensive cost of this hardware, it is often not realistic to provide such a regulator for each individual core of a many-core processor; instead, chip manufacturers group them into islands and energy optimizing solutions must take this constraint into account when assigning frequencies to jobs. We extend Crown Scheduling in order to compute correct schedules regarding islands constraints and we find that for most task sets, our Crown Scheduler finds almost equally good schedules for target architectures with and without island constraints.

1. INTRODUCTION

The race toward high performance and energy-efficiency has led to several shifts in the design of computer systems and the way to program them. One alternative to increase processing power at a sustainable energy consumption level lies in multicore and manycore processors. However, the design of efficient parallel programs is notoriously more complex than sequential ones and increases with the number of cores. Stream programming provides an abstraction that allows the expression of programs processing a large amount of data by communicating tasks running repeatedly. Parallelism complexity is reduced to the parallelization across tasks, and the overall application throughput is very dependent on *scheduling*, i.e., core allocation, mapping and running frequency of each task in the streaming application. The optimization problem to solve by the code generator targets typically the minimization of either *makespan* (throughput maximization) or *energy*. Also, scheduling problems may involve sequential tasks only, *Moldable*[1] tasks, or *Malleable* tasks. Moldable and malleable tasks can run on one or several cores in parallel, but only malleable tasks can increase or decrease at runtime the number of cores running them. For example, Fan et al. [2] give a survey and propose a 2-approximation of a scheduler of Moldable tasks that optimizes for throughput but ignores DVFS capabilities. To the best of our knowledge, only few techniques take profit of DVFS to optimize energy consumption of schedules, such as the one proposed by Pruhs et al. [9] for sequential tasks, taking communication costs into account but assuming a continuous set of frequencies admissible by the underlying execution platform.

These techniques typically do not take architectural constraints such as voltage and frequency islands into account when optimizing energy. However, architectures that implement frequency scaling such as the SCC [4] often restrict voltage an frequency scaling to groups of cores instead of individual cores. Liu and Guo [5] proposes an island-aware approximation for sequential tasks; see their paper for more island-aware scheduling techniques. However, they only schedule sequential tasks. Xu et al. [10] propose solutions for moldable and malleable tasks, but they too ignore island constraints. Crown Scheduling [8], co-optimizes core allocation, mapping and frequency scaling at the same time for moldable tasks. It performs better than Xu et al.'s scheduler [10] both in optimization time and energy, in most cases [8]. Crown Scheduling is versatile enough to admit extensions such as *core consolidation* [7], which also takes the energy cost of idle cores into account and switches a core off if possible, i.e., only if a core is not used at all due to the necessary, possibly long switching on and off delays.

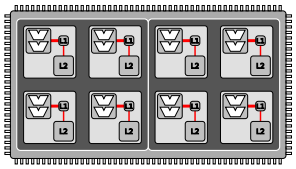
In this paper, we show how to extend our Integer Linear Programming-based Crown Scheduler with core consolidation so that voltage and frequency island constraints are respected. We find that even with these additional restrictions over schedules, our Crown Scheduler produces solutions of almost equal quality than when scheduling for architectures with no island constraints. In the extreme case, comparing a schedule for a target platform with no constraint to another schedule for an equivalent architecture where all cores are grouped in a single island, we find a 40% energy penalty when idle energy dominates the overall consumption and less than 2% when all cores are busy.

2. CROWN SCHEDULING

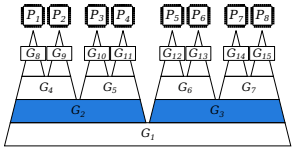
For a concrete machine, it is sufficient to use results from power measurements of the target machine at the available frequencies [1]. We are interested in computing schedules for periodic streaming tasks for energy efficiency, under a throughput constraint. We assume that all tasks are ready at the beginning of each round of the streaming pipeline. We assume further that all tasks have a common deadline M, which is the same as their period.

Crown schedulers partition the set of p identical cores of a target architectures into $2p-1$ disjoint subsets by recursive

[1] A *moldable* and a *malleable* task are tasks that can exploit parallelism to shorten their execution time. A *moldable* task must have its number of cores fixed before the execution of the application



(a) 8-core architecture with a balanced island structure of 4 cores per island.



(b) Balanced crown configuration with groups G_2 and G_3 matching islands shown in Fig. 1a.

Figure 1: Platform with balanced island structure and the corresponding crown configuration for island-aware scheduling. Here $N = \{G_2, G_3\}$.

binary decomposition; see Fig. 1b for an example. A first large subset includes all p cores. This subset is decomposed into two disjoint subsets of $p/2$ cores each, and so on until we obtain p subsets of one core. This leads to a balanced binary tree of processor subsets, where all the cores included in a subset in the tree also belong to its ancestor subset. Melot et al. call such subsets *groups* and the largest subset of p cores the *root group*. Each task is scheduled by mapping it to exactly one of the $2p-1$ core groups and by allocating it all the cores of that group. This reduces the number of possible core allocations from p to $\log_2 p$ for each task, and the number of possible mappings from 2^p to $2p-1 = O(p)$ possible target core subsets and therefore limits considerably the solution search space. All cores execute tasks in the same order by decreasing group size. Furthermore, the recursive decomposition scheme into disjoint subsets eliminates any external fragmentation. As a result, scaling the frequency of tasks mapped to a core group cannot affect tasks of a sibling core group, and no conflicts due to frequency scaling can happen, which simplifies further the scheduling problem.

The set of processors $P = \{P_1, ..., P_p\}$ is hierarchically structured into a set of $2p-1$ *processor subgroups* by recursive binary partitioning as follows: The *root group* G_1 equals P; it has the two child subgroups $G_2 = \{P_1, ..., P_{p/2}\}$ and $G_3 = \{P_{p/2+1}, ..., P_p\}$, four grandchildren groups $G_4 = \{P_1, ..., P_{p/4}\}$ to $G_7 = \{P_{3p/4+1}, ..., P_p\}$ and so on over all $L+1$ tree levels, up to the *leaf groups* $G_p = \{P_1\}$,..., $G_{2p-1} = \{P_p\}$. Unless otherwise constrained, such grouping should also reflect the sharing of hardware resources across processors, such as on-chip memory shared by processor subgroups. Let C_m denote the set of all groups that contain processor P_m. For instance, $C_1 = \{G_{2^z} : z = 0, \ldots, L\}$. Let $p_i = |G_i|$ denote the number of processors in processor group G_i. Where it is clear from the context, we also write i for G_i for brevity.

We call T the set of n tasks to schedule for the target architecture. A task j achieves τ_j work and runs at frequency $f_j \in F$ where F is a bounded, discrete set of frequencies admitted by the target architecture. Furthermore, we can allocate $1 \leq w_j \leq p$ cores to any task j and model the parallel efficiency (e.g. the cost of core synchronization) of task j with $e_j(m) \in (0, 1]$ where $e_j(m) = 1$ when there is no penalty for running task j on m processors. Finally, we call *width* the number of cores w_j alloted to task j. In other words, the execution time of task j is

$$r_j = \frac{\tau_j}{f_j \cdot w_j \cdot e(w_j)} \tag{1}$$

The power consumption of a processor core can be roughly split into dynamic power and static power. Dynamic power P_d is consumed because transistors are switching. This is influenced by the energy needed for a switch, which depends quadratically on the supply voltage (as long as this voltage is far enough from the threshold voltage), how often the switch occurs, i.e., the frequency f within set F of applicable discrete frequency levels, and how many transistors are switching, i.e., on the code executed. There are more influence factors such as temperature, but for simplification we assume that minimum possible voltage for a given frequency (and vice versa) are linearly related, and that for compute intensive tasks, the instruction mix is such that we know the average number of transistors switching. Therefore, ignoring the constants, we get

$$P_d(f) = f^3 \ .$$

Static power is consumed because of leakage current due to imperfect realization of transistors. Static power is both dependent on frequency and on a device-specific constant κ. For simplification, we express it as

$$P_s(f) = f + \kappa \cdot \min F \ .$$

For simplification, we group the power spent when executing code as dynamic and static power; we assume a proportionality factor of 1 and get:

$$P_t(f) = \zeta \cdot P_d(f) + (1 - \zeta) \cdot P_s(f) \ ,$$

where $\zeta \in [0; 1]$ expresses the relative importance of dynamic and static power consumption.

With core consolidation, we model the energy consumption of a schedule as

$$\sum_{j \in T, i \in G, k \in F} x_{j,i,k} \cdot \frac{\tau_j \cdot (P_t(f_k) - P_{idle})}{w_j \cdot f_k \cdot e_i(w_j)} + (p - \sum_m (1 - u_m)) \cdot P_{idle} \tag{2}$$

where $x_{j,i,k}$ is a binary variable set to 1 if task j is mapped to group i and runs are frequency k on w_j cores. We use ILP (Integer Linear Programming) to find a schedule that minimizes energy consumption. In order to set the u_m variables correctly, we define additional constraints. If a core m is used, then u_m must be forced to 1:

$$\forall m \in P, j \in T, k \in F : \sum_{i \in C_m} u_m \geq \sum_{i \in C_m} x_{j,i,k}$$

If any task i is mapped (at any frequency) to a processor group that uses m, then the sum is larger than zero, and hence the right-hand side is less than 1. Multiplying by $1/n$ ensures that the right-hand side can never be less than 0.

Forcing u_m to 0 if core m is not used, is not strictly necessary as an optimal solution will set as many u_m as possible to 0 to minimize energy. Yet we include the constraint in our core-consolidated scheduler:

$$\forall m \in P : u_m \leq \sum_{i \in C_m, j \in T, k \in F} x_{j,i,k} \tag{3}$$

3. ISLAND-AWARE CROWN SCHEDULER

In this section, we take profit of the recursive decomposition of Crown Scheduling to extend our ILP-based integrated implementation with constraints that take voltage and frequency islands into account. As power consumption is mostly dependent on voltage (see Sec. 2), we consider

voltage scaling only and we always use the fastest frequency available for a given voltage level. Therefore we focus on voltage islands only.

For a target architecture of p cores, we consider l sets of cores $H_1 ... H_l$ to model islands, where $\cup_{r=1}^{l} H_r = P$. Also, we have R_m the set of islands that contains processor m; obviously, we have $\forall m \in 1..p : |R_m| = 1$, i.e., such a processor belongs to exactly one island. We define $M_i = \cup_{m \in G_i} R_m$ as the set of all islands any core in group i is a member of. N is the set of groups for which the set of cores matches exactly the union of 1 or more frequency islands (Eq. 4); see Fig. 1b for an example. Finally, O_i is the set of descendant groups of group i following crown decomposition, i.e. the set of smaller groups than i having at least one core in common (Eq. 5).

$$N = \bigcup_{i=1}^{|G|} \begin{cases} i & \text{if} \quad \exists r \in 1..l : H_r = G_i \\ \varnothing & \text{Otherwise} \end{cases} \tag{4}$$

$$O_i = \bigcup_{i' \in G : i' \neq i} \begin{cases} i' & \text{if} \quad G_i \bigcap G_{i'} \neq \varnothing \wedge |G_i| > |G_{i'}| \\ \varnothing & \text{Otherwise} \end{cases} \tag{5}$$

We define the binary variable $s_{i,f}$ and we force it to 1 if at least one task mapped to i or one of its subgroups is *not* set to f (Eq. 6)

$$s_{i \in N, f \in F} \geq \sum_{i' \in O_i} \sum_{j' \in T} \sum_{f' \in F : f' \neq f} \frac{x_{i',j,f'}}{n}, \tag{6}$$

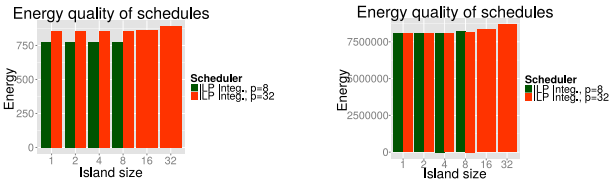
and we make sure that for all group $i \in N$, the amount of frequency levels *not* scheduled for any tasks in i or any of its subgroup, is exactly $|F| - 1$ (Eq. 7), i.e., the number of different frequency levels scheduled for any task in i or one of its subgroup is exactly 1; this is possible only if all tasks mapped to group i or one of its subgroups are scheduled the same frequency level.

$$\forall i \in N : \sum_{f \in F} s_{i,f} = |F| - 1. \tag{7}$$

With Crown Consolidation (Sec. 2), we switch cores off to improve energy consumption. When a core is switched off, its voltage supply is cut to save the associated energy. However, this must be done at the voltage island level, therefore affecting all cores in the island. We add another set of constraints to u_m (see Sec. 2)

$$\forall r \in 1..l, m \in H_l, m' \in H_l : m \neq m' \wedge u_m = u_{m'}.$$

Let us consider a 4 cores target platform with 2 islands of 2 cores each, where the frequency set F is $\{l, h\}$, where $l < h$. We have a task graph of 3 sequential tasks of workload $\tau = h$ and deadline $M = 1$. Since one of the 4 cores is unused, the constraint of Eq. 3 forces our island-aware Crown Scheduler to switch off the other core in the same island; however, this requires to move the task it runs to one of the remaining 2 cores and they cannot fulfill the deadline constraint. In order to make our island-aware Crown Scheduler able to find a schedule for this problem instance, we need to waive constraint 3. This does not affect our scheduler's ability to switch cores off since, as discussed in Sec 2, it would try to switch off as many cores as possible to minimize energy consumption. However, in an island where at least one core runs at least one task, all unused cores cannot be switched off and therefore consume idle energy.



(a) Projected energy consumption for the Concrete task set.

(b) Projected energy consumption for the Streamit task set.

Figure 2: Projected energy consumption for the Concrete and Streamit task sets for platforms of $p \in \{8, 32\}$ cores and islands of 2^q cores with $q \in 0..\log_2 p$ (constant multiple of Joule).

Note that a Crown Configuration is valid for a platform's island topology iff for each island r in the target platform and for each group i, all cores in i are also in M_i and all cores in M_i are also in i, unless all cores in group i belong to a unique island (Eq. 8), i.e., group i is a subset of an island (groups 2 to 11 in Fig. 1b).

$$\forall i \in 1..|G| : G_i = \bigcup_{r \in M_i} H_r \vee |M_i| = 1 \tag{8}$$

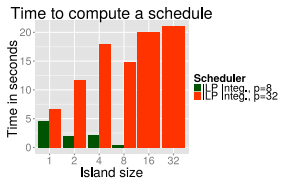
4. EVALUATION

We measure the impact on energy efficiency of static schedule depending on the voltage and frequency island constraints imposed by the target platform. We target several platform variants of $p = 8$ and $p = 32$ cores, where cores are grouped in islands of 2^q cores where q varies between 0 and $\log_2 p$. We compute schedules for applications in a *concrete* task set and a set derived from the Streamit benchmark suite [3]. The concrete task set includes simple classic algorithms such as divide-and-conquer FFT, Mergesort and parallel reduction. See [8] for these applications and how we set their throughput constraint.

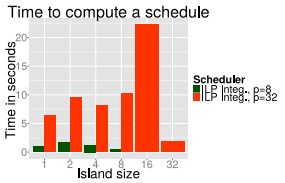
We use AMPL version 20160325 and Gurobi 6.51 with 16 cores Intel Xeon E5-2660 and 32 GiB 1600 MHz memory.

Figure 2 shows the resulting energy for the concrete and Streamit task sets. We see that the quality of solutions computed by our integrated Crown Scheduler decrease moderately with tighter constraints, demonstrating the capacity of our Crown Scheduler to find good scheduling alternatives. For the concrete task set and for a 8 cores platform, there is a marginal energy consumption difference of 0.1%. This difference increases to 4% with 32 cores. For the same number of cores, the Streamit task sets shows a 4% and 7% energy consumption improvement, respectively for 8 and 32 cores.

We observe a maximum of 40% difference in energy quality with application *vocoder* from the Streamit task set between a target platform with no island constraints and with a unique island including all cores. This application involves few sequential tasks and a very loose deadline constraint. When there is no island constraint, our scheduler places all tasks to the same core and switches off all other cores and saves a lot of idle energy. When there is only one island, it cannot switch any core off and they mostly consume idle energy. When the deadline is tight and switching cores off is not possible even with no island constraints, we observe that the possibility of switching frequencies or not, yields a



(a) Optimization time function of island size for our concrete task set.



(b) Optimization time function of island size for our streamit task set.

Figure 3: Optimization time for our concrete and Streamit task sets as a function of frequency island sizes.

maximum energy consumption difference of 2%.

It is interesting to note that in our previous experiments [6] on SCC [4], we observe that computing schedules without taking frequency islands into account and then using the runtime system to set the minimum supply level so that all cores in a voltage island can run at the frequency scheduled, makes the stream program to consume significantly more energy. Here, we see that with a careful scheduling strategy, we can produce results close to the ones we would obtain for compute-intensive task sets with no island constraints and still satisfy the restrictions of a real many-core platform.

We show on Fig. 3a the optimization time for our concrete and Streamit-derived task sets. With 32 cores, we see that more island constraints makes schedulers to run for a longer time; however this is not true for the Streamit task set. We observe the opposite with 8 cores with the concrete task set, where the optimization time decreases with tighter island; with the Streamit task set, the optimization time peaks for intermediate island sizes. However, the optimization time for 32 cores is longer than for 8 cores.

The optimization time increases with the number of variables necessary to model the target architecture, whereas cutting the search space with more constraints does not necessarily decrease it. For example, when frequency islands allows the switching off of cores, the ILP solver can prune very fast solutions that involve many cores to be active and explore a much smaller solution space to find good mapping and scaling solutions, because this space involves fewer cores. When an unique island allows to switch one or more cores off, the ILP solver can prune the solution subspace where all cores are active and find more quickly the best mapping and frequency setting.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we provide an extension to our Crown Scheduler to take voltage and frequency islands into account. In our tests, the improved Crown Scheduler manages to produces almost equally good solutions for compute-intensive task sets (max 2% penalty). When idle time is dominant in the final schedule, coarse islands make impossible to map all tasks to a unique core and switch others off, resulting in a higher penalty of 40%. There are many other opportunities to take profit of the crown structure. For instance, future work includes crown memory bandwidth scheduling for task sets with heterogeneous memory usage behavior.

Acknowledgments

Partial funding by EU FP7 EXCESS (611183) and SeRC, and by the CUGS graduate school at Linköping University.

References

[1] P. Cichowski, J. Keller, and C. Kessler. Modelling power consumption of the Intel SCC. In *Proceedings of the 6th MARC Symposium*, pages 46–51. ONERA, July 2012.

[2] L. Fan, F. Zhang, G. Wang, and Z. Liu. An Effective Approximation Algorithm for the Malleable Parallel Task Scheduling Problem. *J. Parallel Distrib. Comput.*, 72(5):693–704, May 2012.

[3] M. I. Gordon, W. Thies, and S. Amarasinghe. Exploiting Coarse-grained Task, Data, and Pipeline Parallelism in Stream Programs. In *Proc. 12th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, pages 151–162. ACM, 2006.

[4] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. Van Der Wijngaart. A 48-Core IA-32 message-passing processor in 45nm CMOS using on-die message passing and DVFS for performance and power scaling. *IEEE J. of Solid-State Circuits*, 46(1):173–183, Jan. 2011.

[5] J. Liu and J. Guo. Voltage island aware energy efficient scheduling of real-time tasks on multi-core processors. In *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, 2014.

[6] N. Melot, J. Janzen, and C. Kessler. Mimer and Schedeval: Tools for Comparing Static Schedulers for Streaming Applications on Manycore Architectures. In *Parallel Processing Workshops (ICPPW), IEEE*, pages 146–155, Sept 2015. doi: 10.1109/ICPPW.2015.24.

[7] N. Melot, C. Kessler, and J. Keller. Improving Energy-Efficiency of Static Schedules by Core Consolidation and Switching Off Unused Cores. In I. Press, editor, *Proc. of Int. Conf. on Parallel Computing (ParCO 2015)*, pages 285 – 294, Edinburgh, UK, September 2015. doi: 10.3233/978-1-61499-621-7-285.

[8] N. Melot, C. Kessler, J. Keller, and P. Eitschberger. Fast Crown Scheduling Heuristics for Energy-Efficient Mapping and Scaling of Moldable Streaming Tasks on Manycore Systems. *ACM Trans. Archit. Code Optim.*, 11(4):62:1–62:24, Jan. 2015. ISSN 1544-3566.

[9] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed Scaling of Tasks with Precedence Constraints. *Theory of Computing Systems*, 43(1):67–80, July 2008.

[10] H. Xu, F. Kong, and Q. Deng. Energy Minimizing for Parallel Real-Time Tasks Based on Level-Packing. In *18th Int. Conf. on Emb. and Real-Time Comput. Syst. and Appl. (RTCSA)*, pages 98–103, Aug 2012.