Towards automatic adaptation of resource requirements to actual workload for task-based programming models.

Georgios Varisteas  
KTH Royal Institute of Technology  
yorgos@kth.se

Mats Brorsson  
KTH Royal Institute of Technology  
matsbror@kth.se

ABSTRACT
Given the considerate amounts of research on task-based parallel programming models for maximizing performance, this project tries to go one step further and investigate multiprogrammed many-core systems. We provide a mathematical model for a parallel runtime that automatically adapts the need for resources to the workload actually produced. The programming model developed is based on other well established models like Cilk++ and Wool.

The paper concludes with two conditions that guarantee over-utilization and under-utilization of of the existing workers without having to capture efficiency metrics periodically.

The work presented here is the first step towards a complete framework for the system-wide scheduling and load balancing of multiprogrammed many-core system, assuming a variety of workload types and guaranteeing at least average execution for each running program.

1. INTRODUCTION
This project's main focus is maximizing overall utilization and throughput of a multiprogrammed system. In other words trying to maintain acceptable performance of multiple applications running simultaneously, while resource wastage is kept to a minimum. Resources are typically wasted in such configurations due to deficiencies in programming models that have been designed and tested on dedicated hardware.

The work presented is based on two major assumptions, a distributed operating system [3] and task-based work-stealing programming models.

For successful load balancing there are two necessary ingredients; knowledge of available resources and each application’s effective resource usage. An old idea for gathering this information has been feedback based mechanisms to a centralized system scheduler [4]. Previous experiments stumbled upon a single question, which part of the system can decide optimally for the system-wide allocation of the resources?

Related projects have delegated the full responsibility of that task to a system level scheduler, which aggregates efficiency measurements on all running applications and periodically partitions the resources.

Given that the notion of a period is not well modelled in the target applications, it is our belief that such schemes are inflexible when scaled. The presented model tries to decouple application efficiency judgement and decision on resource distribution. This is achieved by allowing the application to know on its own if it should change its resource allocation. Realization of such a change translates to a request to the system scheduler who in turn can reject it according to current availability. We believe that this scheme is highly flexible.

For the purposes of this project we assume a custom fork-join programming model that is quite similar to Cilk [4] and WOOL [7]. To the semantics of those models we have made two major customizations. The use of persistent shared worker threads, created at boot-time as a single worker per physical core. Second is a complex non-random work-stealing protocol between those workers.

Without any control over the worker threads each application is considered as a simple task-generator and is called a taskset; such a set includes all the tasks to be spawned during the execution of an application. Each taskset is assigned to a subset of the existing workers, forming a work-stealing island. Workers can be shared between islands. All such islands are modifiable in size, meaning a change in the number of workers allowed to process each taskset. The mechanism for deciding any such change is self-contained and consists our main contribution.

Due to the limited size of this paper, several parts of the mathematical foundations for this model (mostly proofs of several lemmas and corollaries) have not been included. However, they can be produced upon request.

1.1 Principles
The model presented is based on three fundamental principles:

1. Island size containment: due to significant latencies occurred from cores interconnect networks, expanding the island should occur only when the existing parallelism allows it, while it should be reduced when there is not enough work produced.

2. Space sharing prioritization: Inter-island sharing of workers should be minimized to the ones on the virtual borders of the islands involved.

3. Malleability Cost minimization: island size malleability is usually frequent. Thus adding and removing workers should be ideally cost free.

1.2 Architectural assumptions
This model is based on a specific set of assumption for the underlying architectures it will be deployed on.

Topology formation: It is assumed that cores form a mesh grid, where interconnects can appear either vertically of horizontally but not diagonally. One example of such architecture is the Tilera platform.
Topological dimensions: It is assumed that the topology can span any number of dimensions $m$, although it is more practical to envision it in 2.

Topological border isolation: It is assumed that there is no direct interconnect between cores placed on opposite edges of the chip.

An immediate corollary of the above assumptions is the number of immediate neighbors $N_i$ for each core $i$:

$$N_i \leq 2m$$  (1)

while it is obvious that for the outer cores the available neighbors are at most half.

2. FORMAL DEFINITIONS

Here are some definitions that are used in defining the mathematical model.

$\| \cdot \|$: Measure is a function over any set in this model which provides the size of that set as a count over the elements it includes.

$W$: Workers set. There is one worker for every available physical core in the underlying architecture. $^1$

$w_i \in W$: a worker $i$

$s$: Task-set, is defined as the virtual set of all the tasks that originate from a single initial task run on a source $S_i$. Practically it corresponds to a single application executed using this model.

$S_i$: the source worker that initiates a task-set $s$. This is not selected randomly and it is perceived as the topological center of control for the dissemination of tasks.

$hc(w_1, w_2)$: distance of a worker from another worker as the shortest-path (hop-count) between their respective cores, according to the architectural assumptions. We write $hc_{i,s}$ (or just $hc_i$ if talking about a single task-set) and mean $hc(t_i, S_i)$.

$d_i$: Diaspora of $s$. Maximum $hc$ of a worker from $S_i$ to be allowed to steal from $s$. Diaspora is used to contain a task-set within a specific island, while it also defines and controls the size of the island. Diaspora is non-periodically malleable according to the requirements of the task-set, in correlation to those of the system.

$T_s$: thieves, defined as the set of workers within a distance $d_i$ from $S_i$. These are the only workers allowed to steal a task belonging to a task-set $s$.

$$T_s = \{ w_i \in W : 0 < hc(w_i, S_i) \leq d_i \}$$  (2)

$I_s$: Island of task-set $s$. This is the set of all workers assigned to process a task-set $s$. It is equal its set of thieves $T_s$ plus the source $S_i$. Thus:

$$I_s = T_s \cup \{ S_i \}$$  (3)

t_i,s: a worker acting as a thief, where $t_i \in T_s$

$B_{i,s}$: Bag of tasks of thief $t_i$ from task-set $s$. Each bag includes only the stealable tasks that were spawned by worker $w_i$. Conceptually each worker has a separate bag for each island it belongs to. Tasks that enter a bag are called stagnant tasks.

$Z_c(w_i)$: Distance Zone from $w_i$. This defines the set of workers that have the same distance from a specific worker $w_i$. Thus:

$$Z_c(w_i) = \{ w_j \in W : hc(w_j, w_i) = c \}$$  (4)

We will say $Z_i$ and mean $Z_{hc}(S_i)$ over task-set $s$, where $0 \leq d_i \leq d_s$. Also, assuming $Z_j$ then we call $Z_{i-1}$ and $Z_{i+1}$ its inner and outer zones respectively. Finally, $Z_1$ and $Z_s$ are the innermost and outermost zones of the island $I_i$: $Z_0 = \emptyset$ since no worker can be at 0 distance with any other.

$X_h$ & $X_v$: respectively horizontal and vertical Diaspora control axes or simply axes$^2$. These are defined as the sets of thieves neighboring only one worker of their inner zone, excluding the outermost when $d_s > 1$.$^3$

$$X_s = X_h \cup X_v = \{ T_s' : d_s = 1 \} \setminus \{ T_s' : d_s > 1 \}$$  (5)

$F_s$: Peripheral thieves are the non-outermost ones that are not in the axes, in a task-set $s$.

$$F_s = T_s \setminus (X_s \cup Z_s)$$  (6)

![Figure 1: The boxed workers (excluding $S_i$) are part of the X set. The ones with dashed borders are in $F$](image)

After these necessary definitions it should be clarified that writing worker $w_i$ corresponds to any arbitrary worker in the system ($W$): a thief $t_i$ [in a task-set $s$] is any worker that is part of a specific island $I_s$ and can steal tasks from an active task-set $s$; a bag $B_s$ belongs to the thief $t_i$. When dealing with a single task-set $s$, the indicator $s$ will not be written but assumed, to avoid heavily dense text.

3. TOPOLOGICAL PROPERTIES

According to the topological assumptions and the definitions of the previous section, several properties can be calculated which are necessary for the proofs that follow.

$^1$Available are the cores that are not reserved for the operating system and other functions.

$^2$In the absence of a coordinate system to distinguish between these sets, we mathematically define only their union $X$, as it’s considered trivial for the reader to realize the distinction.

$^3$W: $\{ t_i \in T_s : \exists \forall t_j \in T_s (d_s > hc_i = hc_j + 1 \land hc(t_i, t_j) = 1) \}$
Lemma 1 (Distance zone size). The size of each distance zone $Z_k$ can be calculated in relation to its distance $k$ from the source and the topology dimensions $m$:

$$\|Z_k\| = k^{m-1}2^{m}, \text{ where } 0 \leq k \leq d_s$$

(7)

As a direct result there is a formula for calculating the complete size of the thieves set.

Corollary 1 (Thieves set size). Given lemma 1 the total number of thieves in the island $I_t$ is:

$$\|T_t\| = \sum_{k=0}^{d_s} \|Z_k\| = \frac{(2d_s + 1)^m - 1}{m}$$

Lemma 2 (Zone member distance). Thieves in the same distance zone are at distance greater or equal to 2.

$t_i, t_j \in Z_k \Rightarrow hc(t_i, t_j) \geq 2$

4. THEFT POLICY

This section defines the policy followed by workers for stealing tasks within an island. Moreover, the basic definition is expanded with a series of proved corollaries and other conclusions. This analysis paves the path to understanding and proving the malleability properties of Diaspora in section 5.

4.1 Victims set

Definition 1. Thieves in set $s$ steal from a specific set of workers called the victims set. This set is different for each thief in a task-set $s$ and its definition is given by the conjunction of two rules. First we have the set of thieves of $s$ that are at distance 1 from the thief $t_i$, called $V_{C,i}$. Second, if the thief is a member of the control axes $X_s$ then it has additional victims, defined as all other members of $X_s \cup F_s$ that are at distance 2; this set is empty for all thieves in $F_s$; finally thieves in the outermost zone can steal from thieves in the same zone at distance 2. We write $V_i$ and mean the victims set of thief $t_i$.

$$V_{X,i} = \{ t_n \in Z_s : hc(t_i, t_n) = 2 \}$$

$$V_{C,i} = \{ t_j \in T : hc(t_i, t_j) = 1 \}$$

$$V_i = V_{C,i} \cup V_{X,i}$$

Furthermore, there can be a correlation between the $V_{C,i}$ and $V_{X,i}$, and the distance zones of each victim as defined by the following definitions.

Definition 2. A thief’s $V_{C,i}$ set can be split into tow sub-sets according to the distance zone of each thief. The first is the Outer victims set $O_i$ consisting of the victims in the outer zone of the thief. The second set includes all remaining victims and thus is not defined implicitly as it equals the subtraction of $O_i$ from $V_{C,i}$. Assuming thief $t_i \in Z_1$.

$$O_i = \{ v_j \in V_{C,i} : v_j \in Z_{j+1} \}$$

The purpose of $O_i$ becomes apparent when thought as the set of outer to $t_i$ thieves, that steal from $t_i$. In other words it’s those workers that may look at $t_i$ for work.

It is obvious that $V_{C,i} \cap V_{X,i} = \emptyset$ due to lemma 2.

Corollary 2. Assuming thief $t_i \in Z_j$, all members of its $V_{X,i}$ set are members of the same distance zone.

The reason for enforcing this rule is to contain the size of islands to the necessary amount of workers, as required by the overall workload of the task-set; moreover, it provides higher control over the dissemination of tasks and certain guarantees, which are described and explained later on.

5. MALLEABILITY OF DIASPORA

As mentioned earlier, the islands defined in this model are self managed. This means that the size of the island can be increased or decreased automatically by evaluating specific conditions, called the Diaspora Malleability Conditions (DMC). Increase happens when it is identified that the amount of produced work is enough to utilize more workers, while the size decreases when border workers are found underutilized.

Conjecture 1. The conditions are as follows:

- $d_s$ is increased when the size of the bag of each thief of $s$ in $X_s$ increases beyond $L$.

$$d_+ \iff \|B_i\| > L \geq \|O_i\|, \forall v_i \in X_s$$

Where threshold $L$ will be defined later on.

- $d_s$ is decreased when for all thieves of $s$ that are at distance $d_s$ from $S_i$, their bag is empty.

$$d_- \iff \|B_i\| = 0$$

6. DMC PROOF

It is important to showcase and later prove how these conditions reflect a change in the workload which can justify adding or removing workers from an island. Before doing so, it will be helpful to interpret section 4 from this perspective.

6.1 Workload flow

According to the rules of stealing and how the victims set is constructed, from section 4, there is a specific flow of workload among the thieves in an island. It is imperative for the validation of this model to identify its laws. This will be done here, first descriptively with a generalized example platform and then formally stated.

Assuming a 2-dimensional topology, with a single running task-set $s$, a source $S_i$ and diaspora $d_i = 0$. This is the case for single-core execution in this model.

By increasing diaspora to 1, at maximum 4 more thieves are added into the island (1). These are all at distance 1 from the source and at distance 2 from each other. Since all are in $X_s$, they steal tasks from the source and each other; the source worker can also steal from any of them. Hence, the workload is quickly disseminated and shared among all workers. Although these workers maintain the same behavior with all higher values of $d_s$, some new thieves behave much differently.

Increasing diaspora to 2, will increase the island by 8 more thieves (1), totaling 12 (1) plus the source. An example is given in figure 1, going up to $d_s = 3$. The old thieves of $Z_1$ maintain the aforementioned behaviour. However, all the thieves that have distance 1 from the source are now considered border workers and can steal from each other. This new set of thieves can add more to the overall workload of the task-set; moreover, it provides higher control over the dissemination of tasks and certain guarantees, which are described and explained later on.

Depending on the implementation this can be done either by the island itself or an external dedicated thread.
new thieves are in $Z_s$ and since $d_s < 3$, their $V_{X,s}$ is empty; thus they can steal work from inner thieves but not each other. So, one can say that their purpose is to just lighten the load in $Z_i$.

Assuming that the load increases and condition (8) is met, diaspora is increased to the value $g_3$. At that point $X_s$ is extended to include a subset of $Z_2$, while the rest of $Z_2$ populates set $F_s$; all new thieves are in $Z_s$ but now their $V_{X,s}$ set is not empty. The flow of workload in this scheme is very different. The new members of $X_s$ at $Z_2$ can steal tasks from $F_s$ while the latter workers steal from $Z_1$. Thieves from $Z_2$ will create a new ring distributing the workload, while the main purpose of the $F_s$ thieves is to reflow it inwards (due to prioritization).

If the existing workload justifies the increase of $d$, then new tasks will be stolen outwards, allowing for the peripheral thieves to migrate it back inwards. Thus the dissemination of work is balanced. If the load did not justify the increase, i.e. there are multiple task-sets bordering or even overlapping one another, the thieves of $Z_s$ will be able to steal work from them thus negating the bad estimation. Eventually, as well as in the case of a single task-set, diaspora will quickly decrease again.

To prove that the DMC conditions are correct it is required to prove their necessity and sufficiency. The first part is straightforward, just show that when each condition is true there is indeed an increase or decrease in workload enough to utilize one more or less zone of workers. For that it is necessary to add a few new definitions:

**Definition 3. Task generation rate.** $\frac{\Delta G_i}{dt}$ is the rate at which a worker spawns new tasks in an island $I_i$. We write $\frac{\Delta G_i}{dt}$ for the overall rate in the axes set $X_i$ or $\frac{\Delta G_i}{dt}$ for the whole island. In case of referring to a different set the notation will be $\frac{\Delta G(A)}{dt}$.

**Definition 4. Workload.** $Load_i$ is the maximum amount of stagnant tasks of a thief $t_i$ in an island $I_i$ and is defined as its task generation rate minus the number of its outer victims. Thus:

$$Load_i = \frac{\Delta G_i}{dt} - ||O_i|| \Rightarrow ||B_i|| \leq \frac{\Delta G_i}{dt} - \sum_{j \in O_i} \left( \frac{\Delta G_j}{dt} > 0 \right) : 1$$

We write $Load_A (A$ being a set) and refer to all members of that set individually.

**Definition 5. Thief’s work potential.** $\frac{\Delta N_i}{dt}$. This corresponds to the potential of a thief to have work and is defined as its own task generation rate plus the load of its victims set.

$$\frac{\Delta N_i}{dt} = \frac{\Delta G(t_i)}{dt} + Load_V_i$$

At this point it becomes obvious that a positive value for $Load_i$ denotes an excess in workload, while a negative a lack of it. Furthermore, a positive work potential for the outermost thieves speaks towards the workload stability of the island. This conjecture stems from the stealing scheme that creates a dependency for these thieves to their inner zone. To elaborate, a high such value means that either they can produce enough work themselves or do their inner victims.

### 6.2 Workload increase

The first step in proving the validity of condition (8), is to show that when the condition is true there is indeed an increase in workload enough to utilize one more zone of workers.

It is obvious that the given condition should translate in an increase in workload which can be expressed as follows:

$$\|B_i\| > L \geq ||O_i||, \forall v_{i} \in X_s \Rightarrow \forall w_{j} \in I_s : Load_j > 0$$

**Proof (part 1: sufficiency).**

Assume $\|B_i\| > ||O_i|| \Rightarrow Load_i > 0 \forall v_{i} \in X_s$

$$\Rightarrow \frac{\Delta G_i}{dt} > ||O_i|| \forall v_{i} \in X_s$$

Now assume that: $\forall w_j \in I_s \setminus X_s \Rightarrow Load_j \leq 0 \Rightarrow \frac{\Delta G_j}{dt} \leq ||O_j|| \forall w_j \in I_s \setminus X_s$

However, due to how an island is constructed it holds that:

$$\forall w_j \in I_s \setminus X_s \exists w_i \in X_s : w_j \in O_i$$

This last part contradicts the initial hypothesis which guarantees the existence of stealable work for all $w \in W$. Furthermore, $\forall w_j \in (I_s \setminus X_s) \setminus Z_s$, there exist $w_i \in X_s : m \leq i \leq ||W||$ such that $w_j \in O_i$, meaning the availability of excess work as compared to the number of workers in the island.

However, the actual condition speaks of $\|B_i\| > ||O_i||$ so it requires at least one more stagnant task to justify a Diaspora increase. Practically it is left up to the implementer of this model to decide on a value of the $L$ threshold, with a minimum of $\|O_i\| + 1$.

The second and last part of the proof deals with the necessity of the condition. In other words:

$$Load_j > 0, \forall w_j \in I_s \Rightarrow \forall v_i \in X_s : \|B_i\| > L \geq ||O_i||$$

**Proof (part 2: necessity).**

$$\forall w_j \in I_s : Load_j > 0 \Rightarrow \forall v_i \in X_s : \frac{\Delta G_i}{dt} > ||O_i||$$

$$\Rightarrow \forall w_i \in X_s : \|B_i\| = \frac{\Delta G_i}{dt} > ||O_i||$$

### 6.3 Workload decrease

Proving the condition for reducing the value of diaspora is simpler. First it has to be shown that in the absence of excess workload the bags of the $X_s$ set’s workers are empty.

**Proof (part 1: sufficiency).**

Assume $\forall w_i \in X_s : \|B_i\| = 0$

$$\Rightarrow \frac{\Delta G_i}{dt} \leq \sum_{j \in O_i} \left( \frac{\Delta G_j}{dt} > 0 \right) : 1$$

$$\Rightarrow \forall w_i \in X_s : \forall w_j \in O_j : \frac{\Delta G_j}{dt} \leq 0$$

$$\Rightarrow \forall w_i \in I_s : Load_i < 0$$


Proof (part 2: necessity). Due to the rules of stealing, excess work travels outwards while lack of it will concentrate the work inwards (lack of victims for the outermost workers). Hence:

$$\forall w_i \in X_i \Rightarrow \forall v_j \in O_j : \frac{\Delta G_j}{dt} = 0$$

$$\Rightarrow \forall w_i \in X_i : \|B_i\| = \|O_i\|$$

However, lack of work means that $Load_i < 0$ so:

$$\forall w_i \in X_i : Load_i < 0 \Rightarrow \frac{\Delta G_i}{dt} < \|O_i\|$$

$$\Rightarrow \forall w_i \in X_i : \|B_i\| = 0$$

7. RELATED WORK

As this work comes to enhance Barrelfish OS with efficient resource management mechanisms for multicore and manycore architecture, there have been several other projects with the same goals. Some approaches are similar, like the Factored OS [9], Tessellation OS [6] or ROS [8], opting for partitioning of resources and a redefinition of the process model for the new requirements and complexity introduced by emerging architectures. A common point between all these attempts and our proposal over Barrelfish, is the two level split of the scheduler between the application runtime and the system level. The first understands the requirements while the second knows about the availability.

A point of divergence of our work is the focus of responsibilities between those two levels. Although a strict taxonomy and classification of real life mixed workloads is still not available, it is commonly accepted that rapid fluctuation are normal if not expected. For that reason, a centralized decider that tries to fit diverse workloads in the same set of rules is inefficient when large scale systems are considered [2]. It is our understanding that for maximizing the utilization of such systems it is necessary to change the process model. In a task-centric programming model over a distributed OS, the only currencies are the number of tasks and the number of workers processing them. The first can be modeled as the resource requirement while the latter is the actual resource to be brokered between applications. This way it becomes clear that dividing the scheduler should be combined with moving control of the respective roles too; namely moving control of all threading to the system scheduler.

Moving on, the seminal paper on work stealing by R. Blumofe et al. [5] provides an insurmountable foundation on the performance benefits of random victim selection. However, on large scale manycore systems and in a multiprogrammed context, random work stealing can be the reason for lack of control and flexibility. Through randomness comes unpredictability and in our case the inability to control the flow of the workload. In order to accomplish true resource partitioning and also avoid contention even within a single island, it is very important to predictably control the flow of the workload.

8. FUTURE WORK

This model presents and investigates the properties of a configuration of worker threads to adapt to the existing workload produced by a single task-based program. The next step is to extend the model, both in theory and through implementation, for multiple islands running simultaneously. Several aspects of the model are left to the implementer and one of our points of research would be to investigate different strategies. To name a few, island overlapping against forced release of resources, allocation of secondary resources like the off chip memory bandwidth. Our ultimate goal is a model that can guarantee fair resource management for multiple running applications while allowing leeway for per-application performance optimizations.

9. CONCLUSIONS & CONTRIBUTIONS

This project’s contributions are twofold. One one hand there is a mathematical per application scheduling model that allows a lightweight method for the application to know its requirement of processing resources. The second is a novel framework (shared worker threads and taskset islands) that reduces the overhead of system-wide load balancing, by delegating the per-application scheduling to the application itself; this delegation is performed in a suggestive way from the application to the system, allowing the latter to reject requests for resources.

Our preliminary implementation has reached the point of adaptable single islands with performance that is worst although comparable to the standards set by other similar programming models. We are currently testing thoroughly synthetic benchmarks with fluctuating workloads to test the work dissemination capabilities of the model. Unfortunately such measurements where not mature enough for inclusion to this paper but are going to be available in the near future.

10. REFERENCES


