CONCURRENCY CONUNDRUMS
An Ontological Solution?

Celina Gibbs and Yvonne Coady
University of Victoria, Victoria, Canada

Keywords: Concurrency, Pedagogy, Software comprehension.

Abstract: The arrival of a new era of programming, where developers must consider the subtitles of concurrency inherent in modern many-core architectures, calls for a revamping ranging from fundamental pedagogical processes to software development tools. The problem here is twofold: (1) corresponding real-world scenarios, commonly leveraged in pedagogical practices, contain implicit relationships that are significantly harder to explicitly anticipate in complex code-bases, and (2) the growing plethora of parallel programming language mechanisms collectively blur and distort the common core entities and relationships involved.
As a possible solution, this paper proposes a general ontology for reasoning through concurrency conundrums at both high and low levels. The entities and relationships are originally established based on real-world scenarios presented to a group of grade seven students. The ontology is further developed, and implicit relationships revealed, based on an analysis student observations. The ontology is further developed and implicit relationships revealed. The goal of this work is form a basis for cognitive support that will map equally well to both generalized real-world scenarios and detailed code in different programming languages.

1 INTRODUCTION

Current trajectories suggest that future hardware platforms will house thousands of cores. NVIDIA's Tesla 960 cores for less than $10,000, demonstrates the reality of this situation (Nvidia, 2008). Millions of cores are even available, such as IBM's project KittyHawk (Appavoo et al., 2008).

Parallelism introduces critical issues such as resource utilization and contention which had largely been factored out of mainstream development practices for high-level applications executing in a sequential environment. Programmers are now faced with the daunting task of rethinking and relearning what efficient programming is. This situation calls for researchers to rethink pedagogical practices and programming tools.

In this paper we propose the use of a common ontology that maps to parallel problems both at a high level of abstraction for pedagogical support, and at the level of lines of code for program comprehension tools. This approach is intended to demonstrate the subtle complexities of concurrency in real world scenarios, creating a conceptual model of parallelism that translates to the comprehension of parallel software. We derive the entities of the ontology from both abstract problem-based pedagogy and an understanding of parallel programming models to ensure an ontology that aligns with both levels of representation. In this unified form, the mapping of the ontology to either an abstract scenario or a code base supports both comprehension as well as the ability to compare the implications of concurrency in different contexts.

This paper develops and assesses the ontology as follows. Section 2 draws from experiments with educational based activities revolving around real-life parallelism. Results from these case studies form the top level of our proposed ontology: computation and communication. Section 3 details the significant overlap in this coarse grained entity classification, and demonstrates that deepening the ontology not only clarifies the overlap, but supports the emergence of another top level entity: coordination. Finally, we provide a full mapping of this resulting ontology onto the pedagogical activities in Section 4 with conclusions in Section 5.

2 PEDAGOGICAL ANALYSIS

In order to establish a common ontology that applies at the level of common experiences and at the level of implementation, we begin with an analysis at the more abstract level of activities. The following subsections provide the details for the analysis of three activities presented to a group of grade seven students in an attempt to introduce key concepts about concurrency (Gunion, 2009). The activities used real-world and storyline style scenarios to relate to students, including: (1) two people washing a set of dishes, (2) two people and two movie ticket queues and, (3) an altered version of the classic pedagogical, concurrency scenario of the dining philosophers.

Each scenario allows a range of concurrency to be introduced within the possible solutions for completing the activity. Different strategies for solving the problem introduce different levels of complexity in terms of executing a solution, but the core activity remains the same. We refer to the general notion of 'tasks' associated with each activity as *computation*. With the introduction of multiple students able to participate within each activity, some level of *communication* is required between individuals.

The following subsections specifically investigate the ways in which *computation* and *communication* play out within the solution space defined by the students for each of these three scenarios, and some of the associated consequences encountered when concurrency is introduced.

2.1 Dishwashing Scenario

The dishwashing scenario is proposed as a stack of dirty dishes that need to be cleaned, dried and put away by two people. Though the students proposed a variety of solutions for distributing the work, the core computation remained the same, with differing levels of communication in each scenario.

Computation. Washing a dish, drying a dish and putting a dish away, for all of the dishes in the stack, was immediately identified by all students core elements of computation in this scenario. These small distinct pieces of computation, or sub-tasks, combine to make up the larger complete task.

Communication. The exact communication between participants in this scenario becomes solution dependant, but in general students identified a visual communication mechanism that would occur between participants as one person handed off a dish to the next person. While these 'hand off' points varied, the idea that one individual must complete their portion of the 'computation' before the next could take that dish dictated a partial ordering of sub-tasks.

Concept Overlap - Computation/Communication. Though the students seemed to immediately recognize elements of computation versus communication, they are tightly coupled. In fact, the act of completing one computation is a form of communication to a partner in this work.

2.2 Movie Ticket Scenario

The movie scenario was posed as a problem of two friends wishing to purchase tickets for a popular movie at a theatre with two long queues. This was a slightly concocted problem in which communication was limited by lack of visual contact between queues. Students had to come up with solutions that involved sticking together or splitting up, and dealing with the consequences of each.

Computation. The computation in this scenario is simply the act of purchasing a ticket. Depending on the solution, multiple tickets can be bought by one person or a single ticket by each person.

Communication. In cases where students decided the quickest way to get tickets was to split up, they soon realized the associated consequence of possibly buying too many tickets. This *race condition*, coupled with the non-deterministic processing of the respective lines, required them to consider creative ways to communicate beyond subtle visual cues.

Concept Overlap - Computation/Communication. The overlap between computation and communication in this scenario is solution dependant. In the case of students making use of the two ticket queues, computation can performed across the two ticket booths but introduces the need for communication. In the case of the students staying together in one queue, communication is not necessary, but the concurrency is sacrificed.

2.3 Knights & Forks Scenario

This scenario was altered from its original context of philosophers and described as five knights sitting at a round-table with a single fork between each. A knight required acquisition of two forks in order to eat, if they were not eating they were thinking independently. Students were asked to come up with solutions to manage the forks shared between the knights.

Computation. The computation in this scenario can be identified as the actions of eating and thinking- though attempting to acquire a fork is arguably a sub-task, it was not considered to be a first-class computation.

Communication. In this case, the communication is aided by the visual cue of a fork being available for use by a knight wishing to transition from thinking to eating. This communication between one participant attempting to acquire a fork with two adjacent participants is required to allow a knight to eat.

Concept Overlap - Computation/Communication. As was the case with the dishwashing scenario, visual cues (such as an adjacent knight releasing or acquiring a resource) dictated an ordering such that the end of one sub-task could trigger the beginning of another, though this was not a precondition for the sub-task.

In this activity the consequences of concurrency that are particularly problematic came to the forefront. Students encountered issues of *race conditions*, *deadlock* and *starvation* through role playing, and came up with multiple strategies for managing the shared resource to facilitate each participants' ability to eat. Each of these strategies required some form of communication between participants based on the computation (eating, thinking) patterns.

2.4 Summary

These pedagogical activities served to help us identify common ways in which these students thought about problems associated with concurrency. In each case, students immediately identified tasks/sub-tasks, and coupled these with the often implicit communication mechanisms, including visual cues. Table 1 highlights this breakdown for each activity.

Table 1: Linking real-world examples to computation (CMPT) and communication (COMM) core entities.

Entity	Scenario: Dishwashing (DW)	Scenario: Movie Theatre (MT)	Scenario: Knights & Forks (KF)
CMPT	wash, dry, put away	purchase a ticket	eat, think
COMM	visual, event driven	absent, problematic	visual, mutual exclusion

3 DEFINING THE ONTOLOGY

The preliminary case study described in Section 2 supports the need for a more fine-grained classification, to clean up the overlap or intersection between entities as much as possible. We want to retain the high-level conceptual model of computation and communication, as they were useful as a start.

However, to become more precise, we believe a finer-granularity representation is best achieved through a hierarchical structure provided by an ontology. Further, we hope to derive an ontology that would apply interchangeably to both activities and code and in the future, to other representations of parallel solutions such as patterns. Here we propose a more fine-grained breakdown of computation and communication based on insights from parallel programming research combined with observations outlined in Section 2.

3.1 Identifying Entities

In this first stage of deriving the ontology, we use a combination of related work and concrete evidence from the activities to identify key entities involved.

3.1.1 Computation

In terms of pure computation in a solution to a parallelizeable problem, from our observations we have identified two finer-grained key entities: *task* and *sequential*. The task encompasses the actual computation performed in parallel, or its direct invocation, versus the sequential portion, some might consider the limiting factor in Amdahl's law (Amdahl, 2000).

This task entity aligns with the parallel programming model originally identified in embedded applications (Shah et al., 2003) and high-performance computing (Pancake and Bergmark, 1990), in which one of the critical steps in parallelism is 'the division of the application into parallel tasks'. This division of tasks is also how the students naturally developed solutions at the more abstract level of activities such as the dishwashing and movie ticket scenarios.

The sequential portions of codebases vary substantially in terms of the degree of implicit/explicit resource provisioning in the parallel setup. The student solutions to the parallel activities also differed in terms of what would be considered the sequential portion depending on the decisions in terms of parallelism and guaranteed tradeoffs. For example, the decision of whether or not to split across lines at the movie theatre severely impacts the identification and articulation of the sequential portion of the solution.

3.1.2 Communication

In the case of pure data parallelism, in which no inter-worker communication involved pure communication, is primarily through some form of shared memory or shared state. In these situations a good distribution of data and some control over data access highlights the importance of two finer-grained key entities: *data distribution* and *synchronization*.

Data can be distributed through shared memory or placement of the data within a worker's local memory space, whereas synchronization mechanisms tend to communicate through shared state. This finer-grained breakdown of communication aligns with the following characteristics of parallel support mechanisms from (Asanovic et al., 2006): 'distribution of data to memory elements' and 'inter-task synchronization'.

While the activities given to the students were not examples of pure data parallelism, the same issues of data distribution and synchronization came into play in the problems and solutions. The dishwashing scenario being more of a dataflow concurrency used the shared buffer to communicate when a dish was ready for the next task. This concept of a shared buffer or a shared fork as in the Knights and Forks activity introduce tasks that require synchronization around a resource as well as distribution of data. With out the shared state it was not possible to introduce this same issue of synchronization in the movie ticket scenario which was a good example of task parallelism with access to data distributed across the two queues.

3.1.3 Minimizing the Overlap

While we can somewhat logically parcel out the entities within pure computation and communication in both the activities as *sequential*, *task*, *synchronization* and *data distribution*, these four entities do not encompass everything involved in a parallel solution. Our case study identified an overlap that lies in the intersection of computation and communication, specifically identifying an implicit coordination between these two high level entities. From these observations we identify two entities corresponding to this setup required in a solution as *task coordination* and *data coordination*.

While the task and data distribution are very distinct entities, task coordination and data coordination are more tightly coupled, both involving a provisioning of resources:

- *Task coordination* handles resource provisioning primarily for computation, but it must also be communicated to tasks.
- *Data coordination* handles resource provisioning primarily for communication, associated with computation.

These activities demonstrated that there was very little conscious coordination proposed by the students, but as discussed above the overall system contained combinations of computation/communication based elements that introduced implicit coordination. It was observed in (Gunion, 2009) that coordination emerges as the agreed upon protocol in the suggested solutions within the activities. These protocols can either be coordinating access to shared resources (data coordination) or coordinating the execution of separate tasks (task coordination).

Figure 1 illustrates this overlap and begins to develop our ontology for representing these necessary characteristics of parallel applications.

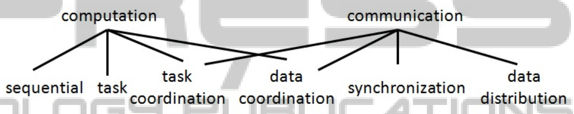

Figure 1: Relationships between computation and communication entities.

Table 2 provides a summary of how each of these six entities could map to a language independent implementation.

Table 2: Mapping fine-grained entities to implementation.

Entity	Implementation Description
sequential (SEQ)	computation coordination of computation
task (TSK)	direct computation computation invocation
task coordination (TC)	resource allocation wrapper functions arguments and context queue management
data coordination (DC)	memory allocation intermediate buffer creation partition function invocation partition size management
synchronization (SYN)	synchronization primitives barriers
data distribution (DD)	data copying partition function data assignment

3.2 Revealing Relationships

These results begin to identify the magnitude of coordination necessary in something as simple as the setup stage in a parallel application. Yet, considering each task on its own—coordination often does not appear to be explicit.

In fact, given the finer-granularity of entities provided in Table 2, we can now re-categorize task coordination, data coordination, and synchronization explicitly to be associated with a new higher-level entity: *coordination*. Here we argue that, though task coordination can be cleanly reassigned to this higher-level, both synchronization and data distribution still contribute to communication as well, resulting in the hierarchy in Figure 2.

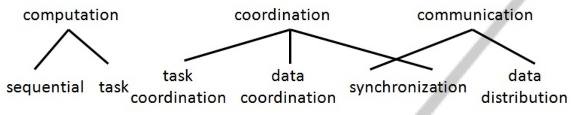

Figure 2: Extended ontology.

This extended ontology of varying granularity supports not only the comprehension of fine-grained entities in isolation, but also provides a holistic view of entity relationships. The key characteristic of coordination entities is their representation of relationships between other entities. These tiers in the ontology provide perspectives that can work in synergy to eventually support the linking of low-level implementation details to high-level abstractions and patterns.

4 FULL ONTOLOGY MAPPING

In this section we reflect back on the details of the educational activities to consider the applicability of the finer-grained mapping proposed in Section 3. Table 3 provides a mapping of the ontology proposed in Section 3 onto the educational activities. The mapping in this table is limited to the assignment of *sequential*, *task*, *synchronization* and *data distribution* to the problem description whereas *task* and *data coordination* are solution specific.

Tables 4, 5 and 6 provide a mapping of task and data coordination to three different solutions for each problem suggested by the students. These tables highlight the differences between the solutions proposed by students participating in the activities describe above. For example, in the movie theatre scenario, the increased level of task coordination required for a distribution across multiple ticket booth becomes evident.

This relationship between the distribution of data and the amount of coordination required to distribute and manage access to it is visible across all three activities. In the dishwashing scenario, solutions DW 1 and DW 2 assign individual tasks to a single person where possible, limiting the amount of shared access to a buffer and therefore requiring less coordination. Similarly, in the Knights and Forks activity, the solutions that eliminate sharing as much as possible (KF 2 and KF 3), require less coordination of resources. The level of complexity of coordination does although have implications in terms of the amount of parallelism possible. This relationship between the coordination and parallelism is visible when considering the coordination in the context of the other entities: sequential, task, data distribution and synchronization. In order to achieve a minimal effort in terms of coordination a tradeoff is made at the level of the task and

Table 3: Full ontology mapping onto activities.

Entity	Scenario MT	Scenario DW	Scenario KF
SEQ	wait in line and purchase a ticket	wash, dry, put away one dish at a time	each knight takes a turn eating
TSK	purchase a ticket	wash, dry put away	eat, think
TC / DC	see Table 4	see Table 5	see Table 6
SYN	not synchronized	wash, dry put away	access to a fork between two knights
DD	ticket booths	sink, rack, counter	between knights

Table 4: Solution specific ontology mappings for movie ticket scenario.

	MT 1	MT 2	MT 3
TC	each person assigned a line, first to front purchases two tickets and communicates when complete	both go in one queue, no coordination, one task	each person assigned a line, purchase one ticket each, meet when complete
DC	ticket access across 2 booths	ticket access only 1 booth	ticket access across 2 booths

Table 5: Solution specific ontology mappings for dishwashing scenario.

	DW 1	DW 2	DW 3
TC	one washes, one dries and both put away	one washes, one dries, both dry and put away	both wash, dry and put away
DC	buffer to hold a washed dish (dryer access) buffer to hold clean dishes (shared access)	buffer to hold washed dishes(dryer access)	buffer to hold washed dishes (shared access) clean dishes (shared access)

Table 6: Solution specific ontology mappings for Knights & Forks scenario.

	KF 1	KF 2	KF 3
TC	coordinate a schedule of: 2 people eating and 3 people thinking	choose who to get rid of and coordinate a schedule of: 2 people eating and 2 thinking	no coordination each person has a set of forks
DC	fork between two knights (shared access)	fork between two knights (shared access)	no coordination each person has a set of forks

data distribution.

For example, in the movie theatre scenario, the task was to wait in line to purchase a ticket while the data was distributed across two booths. The simplest solution in terms of coordination was to make use of only one line up but in looking at the sequential entity for this scenario it is clear that this would sequentialize the solution.

Similarly, with the Knights and Forks scenario, minimal coordination was required in the solutions where a knight was removed or the individual knights took turns eating. Again, the relationship between the task and data distribution entities that is dictated by this minimal coordination can be identified by looking at the entities in Table 3. In this case we see that the task is to eat and think and by giving up one of the knights in fact computational power is being sacrificed. Again, if each night takes a turn eating this a sequential solution as listed in the top row of the Knights and Forks scenario of Table 3.

5 CONCLUSIONS

Currently we face the precarious situation where parallelism is challenging because developers lack a means of exploring possible internal dynamics and causal relationships that tend to be problematic in these code bases. Our study shows that many important relationships are actually concealed or implicit. This is particularly true in real-world scenarios, often used in pedagogical practices when introducing these concepts. Abstractions and mechanisms that hide these relationships as opposed to accentuating them maybe in fact do more harm than good in future code bases.

Though parallelism itself is not a new challenge, the current state of flux for applications and the degree to which they need to be transformed is relatively new and somewhat alarming (Sutter, 2005). The daunting task of efficient programming for highly parallel systems is currently receiving much attention from several perspectives within computer science (Asanovic et al., 2006). This offers an opportunity for researchers to rethink programming models, system software, and hardware architectures from the ground up.

This paper proposes an ontology that would map to the general conceptual entities of a solution to a parallel problem. The entity identification, based on a top-down analysis of conceptual activities was corroborated through consideration of parallel research and existing mechanisms. The application of the ontology onto solutions at the level of activities demonstrates its support for reasoning about and comparing solutions in an aim to convey the tradeoffs within a parallel solution space.

Future work requires a more exhaustive list of these relationships with advice from experts in parallel application development. We also look to ontology experts to help us better define the ontology in order to take full advantage of reasoning engines and cognitive support.

REFERENCES

Amdahl, G. M. (2000). Validity of the single processor approach to achieving large scale computing capabilities. *Readings in computer architecture*, pages 79–81.

Appavoo, J., Uhlig, V., and Waterland, A. (2008). Building a Global-Scale Computer. *SIGOPS Operating System Review*, 42(1):77–84.

Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., and Yelick, K. A. (2006). The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

Gunion, K. (2009). FUNdamentals of CS: Designing and Evaluating Computer Science Activities for Kids. Master's thesis, Department of Computer Science, University of Victoria.

Nvidia (2008). The Nvidia Tesla Supercomputer. http://www.nvidia.com/object/personal_supercomputing.html.

Pancake, C. M. and Bergmark, D. (1990). Do parallel languages respond to the needs of scientific programmers? *Computer*, 23:13–23.

Shah, N., Plishker, W., and Keutzer, K. (2003). NP-Click: A Programming Model for the Intel IXP1200. In *2nd Workshop on Network Processors (NP-2) at the 9th International Symposium on High Performance Computer Architecture (HPCA-9), Anaheim, CA*.

Sutter, H. (2005). The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3).