Towards Data-Parallel Skeletons for Grid Computing: 
An Itinerant Mobile Agent Approach

Seng Wai Loke
School of Computer Science and Software Engineering
Monash University
Caulfield East, VIC 3145, Australia
swloke@csse.monash.edu.au

Abstract

We present an approach to using the skeleton paradigm for grid computing, where the skeletons are executed by mobile agents. The skeletons we use are based on the Bird-Meertens Formalism, involving higher-order operations over data types.

1. Introduction

The mobile agent paradigm utilises software entities that can move from host to host to perform computations [2], and the term agent is a convenient metaphor for a software system that exhibits autonomous and flexible (e.g., proactive, adaptive, etc) behaviour, and have some decision-making (or reasoning) mechanism. The approach for Grid computing we propose is two-pronged: (1) the use of skeletons to express application-specific semantics at a high level, and (2) the use of multiple mobile agents (in cooperation with centralized Grid execution management) to execute the skeletal programs on the distributed Grid infrastructure, managing the low level execution details.

A question now is: how do we map skeletal programs to mobile agents? This paper proposes a solution to this question by presenting a systematic means to map skeletons to mobile agent itineraries expressed in the notation developed in [2]. Our mapping forms a basis for compiling skeletal programs into the mobile agents that will execute the programs.

2 Data-Parallel Skeletons

The data-parallel skeletons we consider in this paper are higher-order functions defined on lists. We only consider two such functions in this paper but there are many more (e.g., see [1]). We write f x to stand for function application, i.e. for f(x).

map. Map, denoted by *, applies a unary function f to every element of a list:1

\[ f * [x_1, \ldots, x_n] = [f x_1, \ldots, f x_n] \]

reduce. Reduce, denoted by /, combines the elements of a list by successively applying a binary associative operator ⨿:

\[ \oplus / [x_1, \ldots, x_n] = x_1 \oplus \ldots \oplus x_n \]

3 A Notation for Mobile Agent Itineraries

Itineraries (denoted by \( I \)) are formed using the following operators. We assume that all agents in an itinerary have a starting place (which we call the agent’s home). Given an itinerary \( I \), we shall use \( \text{agents}(I) \) to refer to the agents mentioned in \( I \).

Agent Movement (\( A^a_p \)). \( A^a_p \) means “move agent A to place p and perform action a”.

Parallel Composition (“||”). Two expressions composed by “||” are executed in parallel. For instance, \( (A_p^a || B_q^b) \) means that agents A and B are executed concurrently. Parallelism may imply cloning of agents. For instance, to execute the expression \( (A_p^a || A_q^b) \), where \( p \neq q \), cloning is needed since agent A has to perform actions at both \( p \) and \( q \) in parallel. When cloning has occurred, decloning is needed, i.e. clones are combined using an associated application-specific operator. For example, given the expression \( (A_p^a || A_q^b) \bullet A^c_{pq} \) and suppose that after the parallel

\[ f * [x_1, \ldots, x_n] = [f x_1, \ldots, f x_n] \]

\[ \oplus / [x_1, \ldots, x_n] = x_1 \oplus \ldots \oplus x_n \]

\[ (A_p^a || B_q^b) \]

\[ (A_p^a || A_q^b) \bullet A^c_{pq} \]
operation, the configuration has clones. Then, decloning is carried out before continuing with $A_1$. The resulting agent from decloning resides in the original place (in this case $s$).

**Sequential Composition (“•”).** Two expressions composed by the operator “•” are executed sequentially. For example, $(A_1^p \cdot A_2^p)$ means move agent $A$ to place $p$ to perform action $a$ and then to place $q$ to perform action $b$.

For the discussions which follow, we assume that the agent (say $A$) has a state variable $r$ (denoted $A.r$) which stores the intermediate results of skeletal computations carried by the agent.

4 From Skeletons to Itineraries

In this section, we show how to convert the higher-order skeletons into itineraries for mobile agents. Below, we use $S \sim A^p \cdot I$ to denote the skeleton $S$ is executed by mobile agent $A$ (and possibly its clones) using itinerary $I$ on the (set of) hosts $P$. For simplicity of discussion, we assume that each element of a list is mapped to a host. Of course, for actual Grid computing, the granularity is much more than a single element of a list.

We denote hosts by the letters $p_1, p_2$, etc., where $p_i \in P$. We use the notation $t[f x]$ to represent task $t$ effectively applies function $f$ onto $x$.

**itinerary for map:** For map, agents are cloned but operate on different data and are sent in parallel to perform the tasks:

$$f \ast [x_1, \ldots, x_n] \sim A_1^p \cdot \ldots \cdot A_n^p$$

where $\odot$ is an agent combination operator.

**itinerary for reduce:** For reduce, agents compute binary operations in parallel, and move as needed to merge intermediate results. For example, for lists of length four, we have:

$$\odot / [x_1, x_2, x_3, x_4] \sim A_1^p \cdot \ldots \cdot A_4^p$$

$$(A_{p_1}^{i[d \text{ id} x_1]} || A_{p_2}^{i[d \text{ id} x_2]} || A_{p_3}^{i[d \text{ id} x_3]} || A_{p_4}^{i[d \text{ id} x_4]})$$

where we assume $P$ contains the four $p_i$’s, $i$ is an identity operation, and $\odot$ is the agent combination operator that takes into account $\odot$, i.e. we define:

$$A_1 \odot A_2 = A$$

where we require that $\odot$ is such that $A.r := A_1.r \odot A_2.r$.

**itinerary for composition of a map and a reduce:** Given a program of the form $\odot / \circ f s$, one could translate it by composing the itineraries for each higher-order function application. For example, we have:

$$(\odot / \circ f s) [x_1, x_2, x_3, x_4] \sim A_1^p \circ A_2^p$$

$$(A_{p_1}^{t[f x_1]} || A_{p_2}^{t[f x_2]} || A_{p_3}^{t[f x_3]} || A_{p_4}^{t[f x_4]}) \bullet$$

$$(A_{p_1}^{i[d \text{ id} x_1]} || A_{p_2}^{i[d \text{ id} x_2]} || A_{p_3}^{i[d \text{ id} x_3]} || A_{p_4}^{i[d \text{ id} x_4]})$$

where $\bullet$ acts as a barrier synchronization point. A more efficient scheme would be to merge the computations and remove the synchronization point:

$$(\odot / \circ f s) [x_1, x_2, x_3, x_4] \sim A_1^p \circ A_2^p$$

$$(A_{p_1}^{t[f x_1]} || A_{p_2}^{t[f x_2]} || A_{p_3}^{t[f x_3]} || A_{p_4}^{t[f x_4]})$$

5 Conclusion

The main benefits of our approach for Grid computing are fourfold: (1) the use of skeletons provides a high-level of abstraction in which to express parallelisable applications; (2) some aspects of management can be removed from global resource managers and delegated to the individual agents; (3) the mapping from skeletons to mobile agents is done systematically, though not formally - we have shown (informally) that the itineraries do indeed implement the intended semantics of the skeletons (and hence of skeletal programs); and (4) our approach facilitates cost analysis and performance prediction of applications since skeletons are translated to prescribed itineraries and itineraries have structure amenable to cost analysis. For example, a time cost function on itineraries is as follows:

$$\text{cost}(A_1^p) = \text{cost}(a) + \text{cost}((\text{move}(\text{here}, p)))$$

$$\text{cost}(I \parallel J) = \text{cost}(\text{cloning}) + \max(\text{cost}(I), \text{cost}(J)) + \text{cost}(\odot)$$

$$\text{cost}(I \bullet J) = \text{cost}(I) + \text{cost}(J)$$

where $\text{move}(\text{here}, p)$ is the cost of moving the agent to $p$, $\text{cost}(\text{cloning})$ is the cost of cloning the agent, and $\text{cost}(\odot)$ is the cost of combining the agent and its clone using $\odot$.

References
