2003 IEEE International Conference on CLUSTER COMPUTING
H O N G K O N G

A Hierarchical and Distributed Approach for Mapping Large Applications onto Heterogeneous Grids Using Genetic Algorithms

Offered By Soumya Sanyal, Amit Jain, Sajal K. Das (University of Texas at Arlington) and Rupak Biswas (NASA Ames Research Center)

Introduction And Motivation

Large Parallel Applications may utilize the power of a heterogeneous Computational Grid as a platform to execute efficiently. Such an efficient execution can be made if the nodes (each represents an atomic task) of the application are assigned to computational resources on the Grid in such a way that effective predicted execution time of the application can be minimized. For large problem sizes in terms of both the application and the Grid, the mapping time becomes a significant factor.

Application, System Models and Problem Formulation

Task Graph : A graph that models a parallel application's computation and communication in terms of nodes and edges.

$G_t = (V_t, E_t)$

$\forall v_i \in V_t, \exists W^{v_i}$ *amount of computation at* v_i *and,*

$\forall (v_i, v_j) \in E_t, \exists C^{i,j}$ *amount of communication between* v_i *and* v_j

System Graph : A graph that models the computational power and the communication cost between nodes in a heterogeneous Computational Grid.

$G_p = (V_p, E_p)$ *where,*

$V_p = \{p_1, p_2, \ldots, p_n\}$ *and* $E_p = \{(p_i, p_j) \mid p_i, p_j \in E_p\}$

$\forall p_i \in V_p, \exists w_{p_i}$ *amount of computational weight and,*

$\forall (p_i, p_j) \in E_p, \exists c_{i,j}$ *amount of link weight.*

The Mapping Problem : "An assignment of vertices of the task graph to the vertices of the system graph so as to minimize the maximum execution time of the busiest processor"

Any mapping F_m *can be defined as ,*

$\forall v \in V_t, p \in V_p, \mu : v \to p$

Let $Exec^{F_m}_{p_j}$ *be the execution time of processor* p_j *when task node* v_i *is assigned to it under a mapping* F_m*. Hence we can say,*

$Exec^{F_m}_{p_j} = Tcomp[v_i, p_j] + Tcomm[v_i, p_j]$ *where,*

$Tcomp[v_i, p_j] = W^{v_i} \times w_{p_j}$ *and,*

$Tcomm[v_i, p_j] = \sum_{v_i \mapsto p_j} \sum_{v_k \mapsto p_l, p_l \neq p_j} C^{v_i, v_k} \times c_{p_j, p_l}, v_i \mapsto p_j$ *means*

processor v_i *is assigned to task node* p_j

The execution time of the application can be given by :

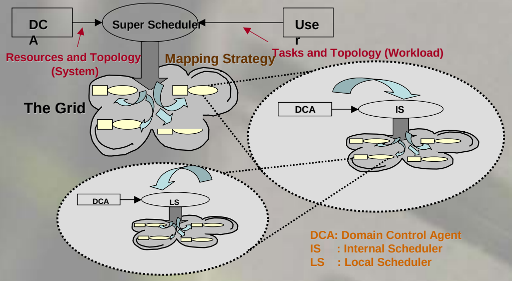
$$Exec^{F_m}_{App} = \max_{p_j}\{Exec^{F_m}_{p_j}\}$$

Therefore, an optimal mapping can be defined to be :

$Exec^{F_m^{opt}}_{App} = \min\{Exec^{F_m}_{App}\} \forall F_m \in MAP, MAP$ *is the set of all possible mappings.*

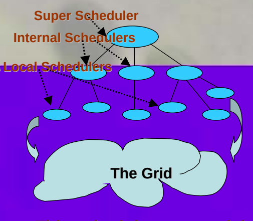
Architectural Assumptions

- **Schedulers (Mappers in our case) are entrusted with mapping a set of tasks to resources within its own domain.**
- **Domain Control Agents supply the set of available set of resources available to the (Super / Internal / Local) scheduler within its own domain.**



Mapping Roadmap

1. User submits job to a scheduler node that becomes the Super Scheduler for that instance.
2. The job is recursively distributed to other schedulers viewed as peers to the Super Scheduler.
3. This is continued further to schedulers under their domains until it reaches the bottom layer of schedulers which map the tasks to compute nodes under it.



This scheduler tree model fits well into a large hierarchical graph that models a network topology.

Our Approach

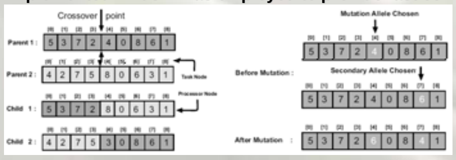
Task Clustering : Contraction **of the task graph to a clustered graph, having smaller number of vertices and edges and consisting of number of tasks equal to the number of *processor clusters.***

Cluster Mapping : Mapping **of the clustered task graph to the available *processor clusters* using GA.**

Recursive Distribution : Recursively **distribute each of the mapped *task cluster* to its respective *processor cluster.***

The GA – Chromosome Encoding and operators -

A simple permutation encoding was chosen and represented by variable length arrays depending on the number of task nodes. The values represented a particular computational node. A simple single point crossover and a permutation mutation operator was implemented. *Elitism* was employed to provide fast convergence.


Crossover point
Parent 1 : 5 3 7 2 4 0 8 6 1
Parent 2 : 4 2 7 5 8 0 6 3 1
Task Node
Processor Node
Child 1 : 5 3 7 2 8 0 6 3 1
Child 2 : 4 2 7 5 3 0 8 6 1
Mutation Allele Chosen
Before Mutation : 5 3 7 2 4 0 8 6 1
Secondary Allele Chosen
5 3 7 2 4 0 8 6 1
After Mutation : 5 3 7 2 6 0 8 4 1

An index above represents a task node number and the values a computational node number. N.B. For higher level schedulers, a computational node is viewed as a collection lower level compute nodes managed by a scheduler under them.

Mapping Metrics

System Potential

$\Phi_{sys} = \sum_{p \in V_p} \Phi_p$

Where Φ_p *is defined as the individual processor potential.*

$\Phi_p = 1/(e_p + \delta \times c_p)$

e_p *= mean (cost) time by p to compute a task averaged over all vertices of the task graph.*

δ *= average degree of a task graph.*

c_p *= mean (cost) time required to communicate with its neighbors.*

$e_p = (\sum_{p \in V_p} w_p / |V_p|) \times W^v, v \in V_t$

$c_p = (\sum_{p,q \in V_p} c_{p,q} / |E_p|) \times \bar{c}_p$ *and* $\delta = 2 \times |E_p| / |V_p|$

$\bar{c}_p$ *is the mean link weight averaged over all links on p.*

$\therefore \bar{c}_p = \sum_{q \in N(p)} c_{p,q} / \delta_p$

The Execution time of the application

Exec$_{app}$= ET (note: this is arbitrary units)

The Mapping time of the algorithm

MT (Mapping time in seconds)

Standard Deviation of ET

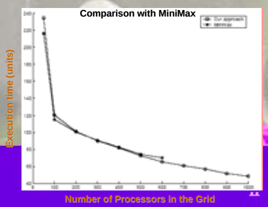
$\sigma = \sqrt{\sum_{p \in V_p} (Exec_p - Exec_{avg})^2 / |V_p|}$

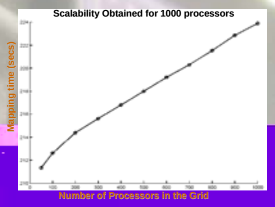
where $Exec_{avg} = \sum_{p \in V_p} Exec_p / |V_p|$

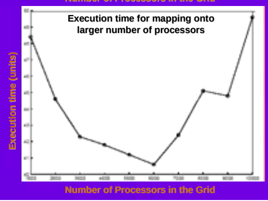
Experimental Results

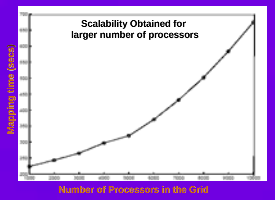
Based on the above discussions, we developed a three-phase algorithm, that :

- **Is distributed : to take advantage of hierarchical scheduler structure.**
- **Clusters the task graph: In a hierarchical tree like fashion to scale down the task graph size and enables distributed mapping.**
- **Maps the tasks : using GA to optimize the mapping using a fitness function that minimizes the maximum execution time of the application.**
- **Is scalable : Testing on a NASA test mesh of 50,000 vertices and system graphs in the order of 10,000 processing nodes.**
- **The Mapping time obtained is sometimes over 100 times faster than MiniMax (another heterogeneous partitioner/mapper).**
- **The Execution time of the application after the final mapping is comparable to MiniMax and outperforms it as the number of system nodes scale in size.**


Comparison with MiniMax
Our approach
MiniMax
Execution time (units)
Number of Processors in the Grid


Scalability Obtained for 1000 processors
Mapping time (secs)
Number of Processors in the Grid


Execution time for mapping onto larger number of processors
Execution time (units)
Number of Processors in the Grid


Scalability Obtained for larger number of processors
Mapping time (secs)
Number of Processors in the Grid