Developing a Cost Model for a Parallel Genetic Algorithm, a Case Study

Mohammad Mahmoud Hamdan *

Received on Jan. 22, 2007 Accepted for publication on Jan. 28, 2008

Abstract

Parallel genetic algorithms are very common in the field of evolutionary computing. One of the methods for parallelizing genetic algorithms is global parallelization. In this paper we discuss how to develop a cost model that can be used for predicting the behavior of a parallel genetic algorithm using a structured top down approach. Other metrics such as speedup, efficiency, isoefficiency and cost-optimality are also developed. The parallel genetic algorithm is used for solving a non trivial combinatorial problem (TSP) and actual measurements are taken. The cost model is used to generate expected performance measurements. The average performance prediction accuracy on a parallel machine is around 96.7% for all experiments.

Keywords: Parallel processing; Performance prediction; Cost models; Parallel genetic algorithms.

Introduction

Genetic algorithms (GA) are part of what is called biologically inspired computing and were developed by John Holland and his colleagues in 1975 [1]. They mimic natural selection and are used for solving search and optimization problems. As we already know, weak organisms within their environment can face extinction by natural selection. During the mating season the fit organisms can pass their genes to future generations. After several generations, only the fit organisms will survive within an environment and they will dominate the entire population. It is possible to develop a computer algorithm that simulates natural selection by generating a random set of solutions, let them evolve in the hope that only the best (and possibly optimal) solution will dominate the entire population after several iterations of reproduction. However, to use a GA successfully it is important to find a suitable encoding of the problem parameters into GA strings and develop an accurate fitness function. Figure 1 illustrates the pseudo-code of a sequential GA.

It is important to note that recent trends in GA are adding parallelism to the behavior of the sequential GA. Hopefully, this will produce better results due to the multiple processors in a parallel machine and reduce the total execution time of the sequential GA. The well known approaches for parallelizing GAs are [2, 3, 4].

© 2009 by Yarmouk University, Irbid, Jordan.
* Department of Computer Science, Faculty of Information Technology,Yarmouk University, Irbid - Jordan.
Hamdan

- Island-models: in this model several GAs are executed in parallel on several processors (called Islands) where the entire GA population is divided among the islands. The islands can have different connection topologies (such as ring, star, … etc). Other issues that are related to this model are: replacement strategy, where to send, what to send … etc.

- Global parallelization: in this model a master node is responsible for sending the individuals to the workers in order to be evaluated in parallel. All other operations such as reproduction and mutation are done on the master node.

However, parallelism adds an additional difficulty in predicting the behavior of the parallelized application. It is important to know whether a parallel version of the GA will run in a reasonable time or not. For example, in the global parallelization model, if the evaluation of a single individual is less expensive than sending it to a worker then there is no point in parallelism. Since GAs can be used for different problems it is expected to have different costs for evaluating every individual in the population. Therefore, it is important to develop cost models that can be used to give accurate prediction of the behavior of the parallel code.

1) Encode the problem parameters as GA strings
2) Develop a fitness functions that can be used to evaluate every string in the population
3) Select strings for reproduction using a parent selection technique
4) Apply crossover and mutation to the selected parents in order to generate a new population
5) Evaluate the new population and repeat steps 3-5 until a termination criteria has been reached
6) Decode strings back into problem parameters and report best result found

Figure 1: A sequential genetic algorithm.

Hence, several approaches have been developed for performance prediction of parallel applications such as [5, 6]: analytical models, deterministic complexity analysis and probabilistic queuing analysis. Also, other researchers presented performance modeling languages such as PAMELA [6]. However, most of these approaches require mathematical background (which are sometimes difficult to develop) and considerable deviations are observed when compared to the actual measurements.

The objective of this paper is to show how a cost model can be developed for a parallel genetic algorithm using a top down approach. The cost model for performance prediction is based on performance profiling [7]. It is important to note that no comparison is given to related work since the PGA developed is unique in its features and no other cost models for PGAs was found that are similar to the PGA under study. The rest of the paper is organized as follows: Section 2 describes the parallel genetic algorithm used in this study. The performance cost model is discussed in section 3. Section 4 presents and discusses the results. Finally section 5 represents conclusions and future works.
The Parallel Genetic Algorithm

Global Parallelization is a widely used technique for parallelizing genetic algorithms. We are developing a cost model for a PGA [8] developed by the author of this paper. It consists of two parts: the root worker and the secondary workers. In the PGA we are using a modified version of the common process farm [9, 10]. Assume there are N nodes in the clusters numbered from 1 to N, then node number 1 will be the root worker and the remaining nodes are secondary workers. The root and the secondary workers are described in the following subsections.

The Root Worker

Figure 2 illustrates the pseudo-code for the root worker that is going to be executed only on the root node. It is called root worker rather than master node as it not only controls the behaviour of the processing nodes but also performs work similar to the secondary workers. It is possible for the root worker to process tasks since the common communication bottleneck in standard master slave paradigm has been reduced by the use of collective communication [11] rather than point to point sends and receives.

R1. Generate initial population randomly.
R2. Evaluate initial population.
R3. Apply parent selection operator and generate candidate parents.
R4. Divide the single population into equally-sized number of subpopulations.
R5. Use a scatter operation in order to send to every worker its corresponding subpopulation. Keep one subpopulation for local processing.
R6. Select and apply crossover operator to the local subpopulation.
R7. Select and apply mutation operator to the local subpopulation.
R8. Decode strings, apply problem-specific heuristics to the local subpopulation and encode strings.
R9. Evaluate local subpopulation.
R10. Use a gather operation in order to collect the new subpopulations from the slave workers.
R11. Apply an elitism operation on the gathered single population.
R12. Repeat steps 3 through 12 until convergence criteria are met.
R13. Broadcast Termination message to secondary workers

Figure 2: Pseudo-code for the root worker.

The role of the root worker is to generate the initial population where all strings in the population are generated randomly according to the representation system used. Global parent selection is then carried out where parents are selected from the current population. The parents are divided across all processors evenly. Each processor (secondary worker) will get its corresponding piece of the population. The root worker will then select and a crossover operator and a mutation and apply it to its local subpopulation. After collecting the new subpopulations it will repeat the process until the PGA converges. It is important to note that this framework preserves the behaviour of the canonical GA as it performs global selection only on the root worker.
The Secondary Workers

Figure 3 illustrates the pseudo-code for the secondary workers. This code will be executed on all processing nodes except the root node. The role of the secondary workers is to receive its corresponding piece of the big population then perform the required operators on the population. The selected crossover and mutation operators will be applied to all strings of the worker’s subpopulation. Different workers will use different crossover and mutation operators. Once the termination message is received from root worker all secondary workers will stop.

S1. Receive subpopulation from root worker using a global scatter operation.
S2. Select and apply crossover operator to local subpopulation.
S3. Select and apply mutation operator to local subpopulation.
S4. Decode Strings, apply problem-specific heuristic to local subpopulation and encode strings.
S5. Evaluate subpopulation.
S6. Send subpopulation to root worker using a global gather operation.
S7. Repeat steps 2 through 7 until a termination message is received from root worker.

Figure 3: Pseudo-code for the secondary workers.

The Cost Model

The cost model for the program needs to take into account the sequential and parallel versions. Therefore, a top-level model (similar to the work of [12] but we differ in estimating communication time using a different approach) has to be built first and it should define the components it comprises. This model is shown below:

\[
T_{\text{GA}} = \begin{cases} 
T_{\text{S\text{GA}}} & \text{if} \ P = 1 \\
T_{\text{P\text{GA}}} & \text{if} \ P \geq 2 
\end{cases} \quad (\text{Eq. 1})
\]

The sequential version is straightforward. It is simply the computation time multiplied by population size (number of individuals) and number of iterations.

A top level model for the sequential version is given below:

\[
T_{\text{S\text{GA}}} = NG \cdot NS \cdot T_{\text{comp}} \quad (\text{Eq. 2})
\]

where,

- \( NG \) : Number of generations (iterations)
- \( NS \) : Number of strings (population size)
- \( T_{\text{comp}} \) : Execution time for the computation part of the program which include parent selection, crossover, mutation, evaluation and problem specific heuristics.
Developing a Cost Model for a Parallel Genetic Algorithm, a Case Study

**Modeling the Execution Time of the Parallel Genetic Algorithm Code**

To derive a cost model for a parallel genetic algorithm, the PGA's performance must be broken down into its principal components [13] with respect to communication and computation times involved. Figure 4 shows the PGA components that are relevant to the performance model. The components are simply: the selection operation at root worker, a scatter operation, the workers and a gather operation. Combining the principal components yields a general performance model for the parallel genetic algorithm.

The PGA execution model is outlined below:

\[
T_{PGA} = T_{Setup} + NG \cdot (T_{Select} + T_{Scatter} + NS \cdot \max(T_{Comp}) + T_{Gather}) \quad (Eq. 3)
\]

where,

- \( T_{PGA} \): Execution time for the application (line R1 through R13 of the pseudo-code in Figure 2).
- \( T_{Setup} \): Execution time for the generation and evaluation of initial population (lines R1 and R2). This time is measured by inserting timing commands before and after the setup operation directly in the code.
- \( T_{Select} \): Execution time for the selection operator at the root worker (line R3).
- \( T_{Scatter} \): Execution time for the scatter operation (line R5).
- \( T_{Comp} \): Is the computation time for all operators involved at the workers and it has to be the maximum computation time among all workers.
Hamdan

- $T_{\text{Worker}}$: Execution time for a worker (line S1 through S6 of the pseudo-code in Figure 3). Note that we take the maximum of all workers including the root worker.
- $T_{\text{Gather}}$: Execution time for the gather operation (line R10).

Modeling the PGA Computation Components

To continue building the cost model for the PGA, it is important to look at the components that were identified in the previous step. The well known approach for estimating the computation time is to measure the time taken to compute one data element then multiply this time with the total number of elements. Therefore, the computation time can be defined below:

$$T_{\text{comp}} = \left[ \frac{T_{\text{cross}} \cdot P_{\text{cross}} + T_{\text{mut}} \cdot P_{\text{mut}} + T_{h1} \cdot P_{h1} + T_{h2} \cdot P_{h2} + T_{\text{eval}} \cdot P_{\text{eval}}}{P_{\text{cross}} + P_{\text{mut}} + P_{h1} + P_{h2} + P_{\text{eval}}} \right] \quad (\text{Eq. 4})$$

where,
- $T_{\text{cross}}$ the crossover time.
- $T_{\text{mut}}$ the mutation time.
- $T_{h1}$ is time needed to perform heuristic 1.
- $T_{h2}$ is the time needed to perform heuristic 2.
- $T_{\text{eval}}$ is the evaluation time.
- $P_{\text{cross}}$ is the crossover probability.
- $P_{\text{mut}}$ is the mutation probability.
- $P_{h1}$ is heuristic 1 probability.
- $P_{h2}$ is heuristic 2 probability.

All probabilities are set by the user and determine how often a given operation will be performed over the population strings.

Modeling the PGA Communication Computation

The last part needed in order to finalize the top-level model for the PGA is to model the communication part in the program. In traditional parallel programs it is common to use standard send and receive operations. The cost model for such kind of communication is simply:

$$T_{\text{comm}} = T_{\text{data}} + \text{SizeOfData} \cdot T_{\text{send}(i,j)} \quad (\text{Eq. 5})$$

where,
Developing a Cost Model for a Parallel Genetic Algorithm, a Case Study

- $T_{\text{comm}}$: Communication time for sending a data structure from one processor to another.
- $T_{\text{init}}$: Initialization and startup costs due to sending a message.
- $\text{SizeOfData}$: The total number of bytes in a given data structure. However, this requires a flattening operation and its cost to be added to equation 3.
- $T_{\text{send}(i,j)}$: Time to send one byte from processor $i$ to processor $j$.

However, this is an old model and the PGA under discussion uses a new model for communication called collective communication [11] (scatter and gather operations). The cost model for the scatter and gather operations is as follows:

$$T_{\text{scatter}(x,y)} = a + by + cx + dx^2 + fc^3 + ey + gxy + hx' + iy' + jx'' + ky'' y$$

$$T_{\text{gather}(x,y)} = a + by + cx + dx^2 + fc^3 + ey + gxy + hx' + iy' + jx'' + ky'' y + l$$  \hspace{1cm} (Eq. 6)

where,
- $T_{\text{scatter}(x,y)}$: the scatter time
- $T_{\text{gather}(x,y)}$: the gather time where $x$ and $y$ stand for number of processors and number of genes (cities for TSP problem) in a string respectively.

It is important to note that this model takes into account the number of processors and cities. The only difference between both equations is in the values of coefficients $a, b, c, d, f, g, h, i, j, k, l, A, B, C, D, E, F, G, H, I, J$ (shown in Appendix 1). Curve fitting technique (the lowest sum of squared absolute error) was used to generate both equations. Actual communication measurements were taken for scatter and gather for different number of processors and number of genes in a string (shown in Appendix 2). The results of the measurements were used to generate the equations shown previously by finding a curve that fits the data.

Combining the Components

To get the complete cost model for the PGA we need to substitute the equations for computation and communication (Eq. 4 and Eq. 6) in Eq. 3. Hence,

$$T_{\text{total}}(p,n) = T_{\text{setup}} + NG * (T_{\text{scatter}} + a + bn + cp + dpn + ep^2 + fp'n + gpn + hp'n + ip' + jp'y + kpn + lpn + n) + NS * (T_{\text{gather}} * P_{\text{start}} + T_{\text{mut}} * P_{\text{mut}} + T_{\text{crossover}} * P_{\text{crossover}} + T_{\text{eval}} * P_{\text{eval}} + T_{\text{mut}} * P_{\text{mut}} + T_{\text{crossover}} * P_{\text{crossover}}) / p)$$  \hspace{1cm} (Eq. 7)

where,
- $p$: is the number of processors.
- $n$: is number of genes in the string.
Evaluation

The parallel genetic algorithm is being tested by solving an NP-complete problem, the travelling salesman problem (TSP). The TSP is a well-known combinatorial problem where a salesman must visit a number of cities exactly once. The aim is to visit all cities with the shortest possible tour in order to minimize travel time and cost. The TSP can be solved using GA but requires special representation (such as permutation encoding) and the fitness function is simply the tour length. However, permutation encoding requires specialized crossover and mutation operators.

The following four anti-symmetric TSP instances: ftv44, ry48p, ft53 and kro124 were taken from TSPLIB [14] and used as input data to the PGA. The instances have 44, 48, 53 and 100 cities respectively. For the experiments we ran the PGA for a fixed number of generations (5000 iterations) for both the sequential and parallel GA. All experiments were conducted on a distributed memory parallel machine. The Linux boxes are connected by a myrinet network (bandwidth of 1 Gbps). Each node has 1 GB of RAM and CPU speed of 3GHz.

Complexity (Asymptotic Analysis)

The complexity analysis of the PGA can be estimated by finding the complexity of all steps involved (lines R3 till R11) in the pseudo-code shown in Figure 2. For brevity we do not show the complete code for the different operations used. The complexity of the PGA is the complexity of the maximum step among the following operations: selection, crossover, mutation, problem heuristics, collective communication and evaluation multiplied by number of generations (NG). The complexity of the maximum operation depends on the encoded problem and type of operators and heuristics used. In this paper, the PGA is used for solving the TSP. Looking at the execution time of the different operations used for the TSP (shown in Appendix 3) we can tell that heuristic 2 has the maximum execution time per iteration. This operator is called the modified Or-opt heuristic. The operator finds all possible subtours of length 1, 2 and 3 cities. The subtour is relocated into all different possible pairs of cities. If the modified tour results in a shorter tour then it is taken. The complexity for this step is $O(\frac{NS \cdot N^3}{P})$ where $N$ is number of cities and $NS$ is number of strings. However, in the PGA, the complete population is divided among all processors evenly which would result in a complexity for the PGA (for useful work) of $O(\frac{NS \cdot N^3}{P})$, where $P$ is the number of processors.

Also there is a complexity for overhead due to communication and synchronization (useless time) which is minimal in the PGA since we are using collective communication. From equation 7, the overhead complexity is $O(K \cdot P^3)$, where $K$ is a constant (very small value) and its value is defined in Appendix 1.2. However, for small to reasonable number of processors we can tell that the overhead complexity is $O(\sqrt{\log P})$ since the
The term \( K \cdot P^3 \) is determined by the value of \( K \) for certain values of \( p \). Therefore, the overall complexity for the PGA is \( O(\frac{N^3 \cdot NN^3}{P} + \log p) \).

**Predicting Execution Time**

Table 1 shows the measured versus predicted execution time (in seconds). For each TSP instance, we report the measured and predicted execution time and the percentage error (ABS(measured time - predicted time)/measured time * 100). The predicted execution time was evaluated by measuring the execution time for the following components (many runs and calculating the average):

\[ T_{\text{setup}}, T_{\text{select}}, T_{\text{cross}}, T_{\text{mut}}, T_{\text{1}}, T_{\text{2}} \text{ and } T_{\text{eval}} \]

and then substituting their values (shown in Appendix 3) in Eq. 7.

The average error for ftv44, RY48P, ft53 and kro124 was about 4%, 3.6%, 4.7% and 1.8% respectively. This evaluation shows very good correspondence between the predicted and measured performance, with an overall average of 96.7% accuracy for all experiments. From the results, it is clear that as we double the number of cities in the TSP instance accuracy improves due to better prediction in computation costs compared with less accuracy in predicting communication costs. This means that for bigger number of cities the total execution times depends heavily on computation costs.
### Table 1: Measured versus the predicted execution time (in seconds) of the PGA

<table>
<thead>
<tr>
<th>#Processors</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftv44</td>
<td>4.9E+02</td>
<td>2.6E+02</td>
<td>1.8E+02</td>
<td>1.4E+02</td>
<td>1.0E+02</td>
<td>8.1E+01</td>
<td>6.0E+01</td>
<td>5.3E+01</td>
<td>4.1E+01</td>
<td>3.7E+01</td>
</tr>
<tr>
<td>ftv44 (Predicted)</td>
<td>4.7E+02</td>
<td>2.5E+02</td>
<td>1.7E+02</td>
<td>1.3E+02</td>
<td>9.9E+01</td>
<td>8.1E+01</td>
<td>6.4E+01</td>
<td>5.4E+01</td>
<td>4.0E+01</td>
<td>3.9E+01</td>
</tr>
<tr>
<td>ftv44 Error</td>
<td>3.4E+00</td>
<td>6.8E+00</td>
<td>4.2E+00</td>
<td>6.1E+00</td>
<td>3.1E+00</td>
<td>3.7E-01</td>
<td>5.2E-01</td>
<td>2.7E+00</td>
<td>2.4E+00</td>
<td>6.3E+00</td>
</tr>
<tr>
<td>RY48P</td>
<td>6.0E+02</td>
<td>3.1E+02</td>
<td>2.1E+02</td>
<td>1.7E+02</td>
<td>1.2E+02</td>
<td>9.4E+01</td>
<td>7.2E+01</td>
<td>5.9E+01</td>
<td>4.7E+01</td>
<td>4.1E+01</td>
</tr>
<tr>
<td>RY48P (Predicted)</td>
<td>5.9E+02</td>
<td>3.1E+02</td>
<td>2.1E+02</td>
<td>1.7E+02</td>
<td>1.2E+02</td>
<td>9.8E+01</td>
<td>7.6E+01</td>
<td>6.4E+01</td>
<td>4.9E+01</td>
<td>4.5E+01</td>
</tr>
<tr>
<td>RY48P Error</td>
<td>1.3E+00</td>
<td>8.6E-01</td>
<td>2.4E-01</td>
<td>1.5E-01</td>
<td>2.5E+00</td>
<td>5.1E+00</td>
<td>5.2E+00</td>
<td>7.4E+00</td>
<td>3.9E+00</td>
<td>9.2E+00</td>
</tr>
<tr>
<td>ft53</td>
<td>7.7E+02</td>
<td>4.0E+02</td>
<td>2.8E+02</td>
<td>2.1E+02</td>
<td>1.5E+02</td>
<td>1.2E+02</td>
<td>8.8E+01</td>
<td>7.1E+01</td>
<td>5.5E+01</td>
<td>4.7E+01</td>
</tr>
<tr>
<td>ft53 (Predicted)</td>
<td>7.7E+02</td>
<td>4.0E+02</td>
<td>2.7E+02</td>
<td>2.1E+02</td>
<td>1.5E+02</td>
<td>1.2E+02</td>
<td>9.2E+01</td>
<td>7.6E+01</td>
<td>5.8E+01</td>
<td>5.1E+01</td>
</tr>
<tr>
<td>ft53 Error</td>
<td>5.8E-01</td>
<td>3.4E-01</td>
<td>3.6E+00</td>
<td>7.4E-01</td>
<td>2.1E+00</td>
<td>3.0E+00</td>
<td>4.0E+00</td>
<td>7.5E+00</td>
<td>6.1E+00</td>
<td>9.4E+00</td>
</tr>
<tr>
<td>kro124p</td>
<td>9.0E+03</td>
<td>4.5E+03</td>
<td>3.0E+03</td>
<td>2.3E+03</td>
<td>1.5E+03</td>
<td>1.2E+03</td>
<td>7.9E+02</td>
<td>6.1E+02</td>
<td>4.3E+02</td>
<td>3.4E+02</td>
</tr>
<tr>
<td>kro124p (Predicted)</td>
<td>8.9E+03</td>
<td>4.5E+03</td>
<td>3.0E+03</td>
<td>2.3E+03</td>
<td>1.5E+03</td>
<td>1.1E+03</td>
<td>7.8E+02</td>
<td>6.0E+02</td>
<td>4.2E+02</td>
<td>3.2E+02</td>
</tr>
<tr>
<td>kro124p Error</td>
<td>1.8E+00</td>
<td>1.2E+00</td>
<td>8.0E-01</td>
<td>2.9E-01</td>
<td>1.5E+00</td>
<td>1.2E+00</td>
<td>1.5E+00</td>
<td>1.6E+00</td>
<td>1.0E+00</td>
<td>7.5E+00</td>
</tr>
</tbody>
</table>
Predicting Speedup and Efficiency

In order to predict program speedup (for \( p \) processors) we need the following:

\[
S_p = \frac{S_{PGA}}{T_{PGA}}
\]  
(Eq. 8)

and to predict efficiency we need the following:

\[
E_p = \frac{S_p}{p}
\]  
(Eq. 9)

Table 2 shows the predicted and measured speedup and efficiency. Since both performance metrics depend on predicting execution time the error is the same for the difference between predicted and measured (not shown for brevity in space). Further analysis is needed in order to understand how efficiency behaves as we increase the number of processors. This analysis is shown in the next section which is called scalability [15] of the PGA.

Table 2: Actual and Predicted Speedup and Efficiency of the PGA

<table>
<thead>
<tr>
<th>TSP Instance</th>
<th># Processors</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftv44</td>
<td>Speedup (Measured)</td>
<td>1.84</td>
<td>2.71</td>
<td>3.38</td>
<td>4.77</td>
<td>5.97</td>
<td>8.05</td>
<td>9.25</td>
<td>11.72</td>
<td>13.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Speedup (Predicted)</td>
<td>1.90</td>
<td>2.73</td>
<td>3.47</td>
<td>4.75</td>
<td>5.79</td>
<td>7.38</td>
<td>8.69</td>
<td>11.60</td>
<td>11.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Measured)</td>
<td>0.92</td>
<td>0.90</td>
<td>0.85</td>
<td>0.80</td>
<td>0.75</td>
<td>0.70</td>
<td>0.58</td>
<td>0.49</td>
<td>0.41</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Predicted)</td>
<td>0.95</td>
<td>0.91</td>
<td>0.87</td>
<td>0.79</td>
<td>0.72</td>
<td>0.61</td>
<td>0.54</td>
<td>0.48</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>RY48P</td>
<td>Speedup (Measured)</td>
<td>1.92</td>
<td>2.79</td>
<td>3.57</td>
<td>5.06</td>
<td>6.39</td>
<td>8.28</td>
<td>10.04</td>
<td>12.72</td>
<td>14.52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Speedup (Predicted)</td>
<td>1.91</td>
<td>2.76</td>
<td>3.53</td>
<td>4.88</td>
<td>6.00</td>
<td>7.76</td>
<td>9.23</td>
<td>12.08</td>
<td>13.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Measured)</td>
<td>0.96</td>
<td>0.93</td>
<td>0.89</td>
<td>0.84</td>
<td>0.80</td>
<td>0.69</td>
<td>0.63</td>
<td>0.53</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Predicted)</td>
<td>0.96</td>
<td>0.92</td>
<td>0.89</td>
<td>0.81</td>
<td>0.75</td>
<td>0.65</td>
<td>0.58</td>
<td>0.50</td>
<td>0.41</td>
<td></td>
</tr>
<tr>
<td>ft53</td>
<td>Speedup (Measured)</td>
<td>1.95</td>
<td>2.71</td>
<td>3.66</td>
<td>5.20</td>
<td>6.54</td>
<td>8.73</td>
<td>10.89</td>
<td>13.97</td>
<td>16.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Speedup (Predicted)</td>
<td>1.93</td>
<td>2.80</td>
<td>3.61</td>
<td>5.06</td>
<td>6.31</td>
<td>8.35</td>
<td>10.07</td>
<td>13.10</td>
<td>15.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Measured)</td>
<td>0.97</td>
<td>0.91</td>
<td>0.91</td>
<td>0.87</td>
<td>0.82</td>
<td>0.73</td>
<td>0.68</td>
<td>0.58</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Predicted)</td>
<td>0.96</td>
<td>0.93</td>
<td>0.90</td>
<td>0.84</td>
<td>0.79</td>
<td>0.70</td>
<td>0.63</td>
<td>0.55</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>kro124p</td>
<td>Speedup (Measured)</td>
<td>2.00</td>
<td>3.00</td>
<td>4.00</td>
<td>5.86</td>
<td>7.76</td>
<td>11.36</td>
<td>14.82</td>
<td>21.08</td>
<td>26.48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Speedup (Predicted)</td>
<td>1.99</td>
<td>2.97</td>
<td>3.94</td>
<td>5.85</td>
<td>7.71</td>
<td>11.33</td>
<td>14.80</td>
<td>20.93</td>
<td>28.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Measured)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.98</td>
<td>0.97</td>
<td>0.95</td>
<td>0.93</td>
<td>0.88</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficiency (Predicted)</td>
<td>1.00</td>
<td>0.99</td>
<td>0.99</td>
<td>0.98</td>
<td>0.96</td>
<td>0.94</td>
<td>0.93</td>
<td>0.87</td>
<td>0.88</td>
<td></td>
</tr>
</tbody>
</table>
Hamdan

Isoefficiency Function and Cost-Optimality

A parallel system can be called scalable [15] if increasing the number of processors improves speedup. However, in many parallel systems increasing number of processors increases communication overhead and thus efficiency decreases for the same problem size. To explain this issue, look at Table 3 that illustrates the relationship between efficiency, number of cities (problem size) and number of processors. For example, to maintain the same efficiency of 0.92 that we got for 2 processors on 16 processors, the number of cities has to increase from 44 cities to 100. To capture this relationship, we need to derive the isoefficiency function [16] for the PGA which is a function of \( p \) (number of processors) that specifies at what rate the problem size should increase in order to maintain a fixed efficiency as \( p \) increases.

Table 3: Efficiency as a function of number of cities and number of processors.

<table>
<thead>
<tr>
<th># Processors</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(number of cities)</td>
<td>44</td>
<td>0.92</td>
<td>0.90</td>
<td>0.85</td>
<td>0.80</td>
<td>0.75</td>
<td>0.70</td>
<td>0.58</td>
<td>0.49</td>
<td>0.41</td>
</tr>
<tr>
<td>48</td>
<td>0.96</td>
<td>0.93</td>
<td>0.89</td>
<td>0.84</td>
<td>0.80</td>
<td>0.69</td>
<td>0.63</td>
<td>0.53</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>0.97</td>
<td>0.91</td>
<td>0.91</td>
<td>0.87</td>
<td>0.82</td>
<td>0.73</td>
<td>0.68</td>
<td>0.58</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.98</td>
<td>0.97</td>
<td>0.95</td>
<td>0.93</td>
<td>0.88</td>
<td>0.83</td>
<td></td>
</tr>
</tbody>
</table>

Therefore, let us call problem size \( W \) and we need to obtain it in terms of \( p \). According to work of Grama et al [16] we have:

\[
W = KT_0
\]  
(Eq. 10)

Where \( K \) is a constant and \( T_0 \) is overhead (extra synchronization and communication time) due to parallelism. As we found earlier in section 4.1 that overhead is \( O(\log p) \). By substituting this value for \( T_0 \) in the equation above we get \( W = K\sqrt{\log p} \). Thus the PGA’s isoefficiency is \( O(\sqrt{\log p}) \). This means that if the number of processors increases from \( p_1 \) to \( p_2 \), then the problem size must increase by a factor of \( \sqrt{\log p_2} / \sqrt{\log p_1} \) to maintain the same efficiency. For example, the efficiency for 44 cities on 2 processors is 0.92. To maintain the same efficiency (or very close to it) on 16 processors we need to increase the number of cities (approximately) by a factor of \( \sqrt{\log 16} / \sqrt{\log 2} \) at least. From these findings it is easy to tell that the PGA is highly scalable due to the big increase in computational complexity as the number of cities increase. Finally, since the PGA's overhead function \( T_0 \) is proportional to the problem size \( W (W \propto T_0) \) then the PGA is cost-optimal. This means that both the overhead function and problem size are of same order of magnitude.
Conclusions and Future Work

We have shown how to develop a cost model for a parallel genetic algorithm using a top down approach. The proposed cost model is built on the work of [12] but refines collective communication time (scatter and gather operations) using a statistical technique thus giving accurate results. The cost model can predict execution time, speedup and efficiency of the parallel GA. Also scalability analysis was performed on the PGA and the isoefficiency function of PGA was derived. The PGA was used for solving an NP-complete problem (TSP). The cost model managed in predicting the execution time of the PGA with a high degree of accuracy. This was shown by the comparison presented in the previous section where measured and predicted performance results were very close.

The model can be used for predicting other instances of the PGA by profiling new operators then substituting their values in the PGA's cost model. For example, using a new crossover operator for TSP (such as heuristic crossover) we can profile the code to find the execution time then use this value for $T_{clos}$ in equation 7.

In future work, it might be useful to use the PGA for solving other problems and then comparing the predicted and measured execution times. Also, the cost model can be generalized so that it can be instantiated with machine specific measurements in order to allow code and performance portability.
References


Developing a Cost Model for a Parallel Genetic Algorithm, a Case Study


Appendix

1.1) Scatter Coefficients

\[ a = -2.5356980741152359 \times 10^{-3} \]
\[ b = 3.5601934318949919 \times 10^{-5} \]
\[ c = 2.1932016914937972 \times 10^{-3} \]
\[ d = -2.3671410553041224 \times 10^{-5} \]
\[ e = -5.065055096365422 \times 10^{-4} \]
\[ f = 6.0397373823080416 \times 10^{-6} \]
\[ g = 4.3715156550506253 \times 10^{-5} \]
\[ h = -5.0072624757863902 \times 10^{-7} \]
\[ i = -1.5627180944908417 \times 10^{-6} \]
\[ j = 2.0295379666027604 \times 10^{-8} \]
\[ k = 1.9584336192415114 \times 10^{-8} \]
\[ l = -2.5895496896198499 \times 10^{-10} \]

1.2) Gather Coefficients

\[ A = -5.2319102532543127 \times 10^{-3} \]
\[ B = 1.0532617362743685 \times 10^{-4} \]
\[ C = 1.2161969673944654 \times 10^{-3} \]
\[ D = -7.670234253823164 \times 10^{-6} \]
\[ E = -2.2058125479655470 \times 10^{-6} \]
\[ F = 2.2344587618615297 \times 10^{-6} \]
\[ G = 2.1337874246005737 \times 10^{-5} \]
\[ H = -2.4815030053933053 \times 10^{-7} \]
\[ I = -9.6194183876231694 \times 10^{-8} \]
\[ J = 1.2135860946673076 \times 10^{-8} \]
\[ K = 1.4938238325674878 \times 10^{-10} \]
\[ L = -1.9868119079954172 \times 10^{-10} \]
The actual measurements used for developing the equations for the scatter and gather communication. For each operation we have taken actual execution time for a different number of processors and cities.

<table>
<thead>
<tr>
<th>Processors\Cities</th>
<th>48</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scatter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.00055</td>
<td>0.000983</td>
</tr>
<tr>
<td>4</td>
<td>0.000898</td>
<td>0.001274</td>
</tr>
<tr>
<td>8</td>
<td>0.000438</td>
<td>0.001761</td>
</tr>
<tr>
<td>16</td>
<td>0.000347</td>
<td>0.000649</td>
</tr>
<tr>
<td>24</td>
<td>0.000368</td>
<td>0.000612</td>
</tr>
<tr>
<td>32</td>
<td>0.000377</td>
<td>0.000833</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gather</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.001136</td>
<td>0.006187</td>
</tr>
<tr>
<td>4</td>
<td>0.001914</td>
<td>0.00698</td>
</tr>
<tr>
<td>8</td>
<td>0.002804</td>
<td>0.008166</td>
</tr>
<tr>
<td>16</td>
<td>0.00379</td>
<td>0.010301</td>
</tr>
<tr>
<td>24</td>
<td>0.002342</td>
<td>0.013897</td>
</tr>
<tr>
<td>32</td>
<td>0.003201</td>
<td>0.007117</td>
</tr>
</tbody>
</table>

The actual measurements for $T_{\text{setup}}$, $T_{\text{select}}$, $T_{\text{cross}}$, $T_{\text{mut}}$, $T_{\text{h1}}$, $T_{\text{h2}}$ and $T_{\text{eval}}$ in seconds for kro124p are (in the evaluation of the model the measurements had to be taken for every single TSP instance):

\[
\begin{align*}
T_{\text{setup}} &= 0.03048026 \\
T_{\text{select}} &= 0.00696298 \\
T_{\text{cross}} &= 0.000020863 \\
T_{\text{mut}} &= 0.00000270851 \\
T_{\text{h1}} &= 0.00000455932 \\
T_{\text{h2}} &= 0.036798219 \\
T_{\text{eval}} &= 0.0000014233
\end{align*}
\]