Distributed Processor Allocation For Discrete Event Simulation and Digital Signal Processing Using a Multiobjective Evolutionary Algorithm

David J. Caswell & Gary B. Lamont
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Wright Patterson Air Force Base, Dayton, OH 45433

Abstract- The use of large scale distributed systems for multiple perhaps heterogenous applications is becoming more commonplace. The organizations that are utilizing these resources must ensure that the applications are executed in a timely manner without unnecessary wasting the resources available on the distributed system. Characteristics of two distributed computing applications are presented; large scale discrete event simulation and a real-time digital signal processing activity. A stochastic processor allocation algorithm is developed for assigning processes to processors in an effective and efficient manner based upon application characteristics. In particular, a multiobjective evolutionary algorithm (MOEA) is created in order to examine Pareto results for such diverse processor allocation. The results indicate that the focus of the two distinct applications and the associated respective optimal regions have distinct differences.

1 Introduction

A distributed system offers researchers the ability to execute applications across several processors. Any given High Performance Computing (HPC) system can have multiple processors working on a single program, or have a variety of programs running simultaneously across the processors. The efficiency and effectiveness of the distributed system is dependent on it's ability to balance the load across all of the processors and communication links. In order to ensure that the HPC system is working effectively for it's intended purpose each application must be examined with regards to the tradeoffs required by the load balancing objectives.

The premise of this paper is to present a multiobjective evolutionary algorithm(MOEA) approach to processor allocation which in general is a NPC problem. Then, using the Pareto multiobjective formulation examine the comparative optimal regions for two different applications. The two systems proposed for this research are: a monolithic discrete event simulation, and a group of multiprocessor signal processing applications. The goal of this research is to allow researchers to understand the Pareto front for these problem classes. With this knowledge allocation systems can be designed such that the *a posteriori* Pareto front selection regions can be coded into the *a priori* aggregate coefficients. This in turn would allow the algorithm to be automated for more practical real-world use.

This document consists of four primary sections. Section 2 explains the two applications that are the focus of this document. Specifically, Section 2.1 describes the signal processing applications and Section 2.2 describes the discrete event simulation with it's potential distributed deployment. Moving to the algorithm that is applied, an explanation of the load balancing model used for this research is provided in Section 3. The specific algorithm and experiments are then described in Section 4 and Section 5 with the multiobjective result comparison between the two applications given in Section 6.

2 Load Balancing Applications

The two algorithms selected for examining the tradeoffs of processor allocation objectives are both Department of Defense based projects that could potentially utilize the load balancing algorithm for improving their overall system performance. The first application that is examined is the digital signal processing work done by the Air Force Research Laboratory (AFRL). It utilizes multiple multiprocessor applications for analyzing signal information. The other application is a large scale simulator used for educating Air Force personnel on doctrine.

2.1 Digital Signal Processing

The analysis of data collected from antenna and sensor arrays is a complex task that must be done extremely fast in order to support the real-time applications that use the data. One project that has been designed to help further the development of HPC antenna/sensor analysis applications is that being done by the Common High Performance Computing Software Support Initiative (CHSSI) with their digital Signal and Image Processing (SIP) efforts. This project examines the use of HPC systems for large scale real-time calculations of the following scalable digital signal processing algorithms:

- Space-Time Adaptive Processing (STAP)
- Multi-Target Tracking/Tracking Toolbox
- 2-D Fast Fourier Transform(FFT)
- M-to-N Data Redistribution

The SIP effort is based on the collection of data from a variety of sensor data including radar, sonar, images, and others. This work supports U.S. military applications including surveillance, reconnaissance, intelligence, communications, avionics, etc. Each of these algorithms have been ported to a parallel environment in order to support the tremendous computational efficiency that real-time processing requires.

2.1.1 Space-Time Adaptive Processing (STAP)

STAP is a method that uses both spatial and temporal information to identify potential targets using statistical analysis. It uses a 2-dimensional FFT filter analysis for detection of targets amongst ground clutter and other types of interference. STAP requires an extensive number of computations in a real-time manner using a data cube[7].

2.1.2 Multi-Target Tracking/Tracking Toolbox

Multi-target tracking system applications are responsible for tracking and differentiating between multiple targets based on the sensor data. Involved in this work is the evaluation for which targets are producing which features from the sensor data. In order to do this analysis multiple filters and sensor data inputs are used with statistical hypothesis generation and testing in order to differentiate between objects [10].

2.1.3 2-D Fast Fourier Transform (FFT)

A fast fourier transform is used to locate signals amongst a large number of interference. It is used not only in sensor analysis but also in quantum physics, linear systems analysis, probability theory and others [13]. Since there are so many applications that take advantage of Fourier transforms a lot of research has been done towards the development of more efficient algorithms. Implementing a 2-D FFT is even more difficult however as each FFT is calculated first in one dimension, then a shift or matrix transpose is performed with the FFT being recalculated in the second dimension. This produces a great deal of both communications and computations making the parallelization difficult an the distribution across processors important for overall efficiency.

2.1.4 M-to-N Data Redistribution

M-to-N Data redistribution attempts to optimize signal processing applications by limiting the number of remote memory accesses needed by each processor. At each phase of the analysis the data are redistributed so that the each processor has the optimal number of data for processing locally [11]. This algorithm works by transforming a cyclically distributed pattern over some initial amount number of processors (M) to a different cyclic on a potentially different (N) number of processors.

All of these applications require a great deal of computing resources provided in a real-time fashion. An effective load balancing algorithm must therefore be able to handle the constantly changing processing requests, allowing for new requests to be quickly inserted into the HPC system without damaging the workload of the processors that are already active.

2.2 Discrete Event Simulation

Discrete event simulations provide users with the ability to imitate a real-world system over a set period of time. By being discrete the system changes only at and for regular time intervals [2]. By using a simulation approach the users are able to, relatively inexpensively, examine the real-world system. This approach also gives users the ability to compress or expand time as needed for analysis. There are a variety of DES systems that model many different real-world applications in everything from computer system networks through restaurant traffic analysis[2]. For the purposes of this paper we singled out a specific training simulation called the Air Force Command Exercise System (ACES). ACES is a synchronized conservative discrete event simulation. It currently is a serial program used for planning with the objectives of aiding users in their understanding and appreciation of a generic simulation:

- Air Force doctrine in a theater exercise.
- The concepts of planning.
- The synergistic effect of integrated air, land, and sea component plans.
- The command and staff relationships involved in combined operations.

The ACES model simulates scenarios for which the users can apply their strategies. In one of the ACES simulations, called Pegasus, seven different simultaneous simulations are performed that involve 90 or more participants. In each of the seven simulations the participants are further subdivided into red and blue teams. These teams then compete against each other using the ACES application.

The simulation moves with simulated 24 hour increments where each team develops a theater campaign plan that they intend to be deployed against their opponents. The simulation itself provides theater maps and status reports while the players determine strategy, logistics and plans.

Once the campaign plan is decided, it is input into the ACES engine. The inputs include the air, land, and sea orders of the units that are resident within the campaign environment. Each of these units is assigned missions that they act during the event step. ACES then proceeds to compute the movements of the forces and the effects thereof for the next 24 hour increment.

Using a distributed model, the ACES system can be expanded into a much larger simulation. Currently, in the serial version all of the players must be decomposed into small subgroups such that each must work autonomously from the other groups. With a distributed version a larger model could be simulated, requiring multiple processors computing scenario regions with events communicated between systems. Thus instead of a small localized region that is independent of anything neighboring, a more real-world model could be implemented. This larger system would provide a better simulation of real-world multi-theater engagements where the regions must contend for the same resources, and actions by one commander could affect the region of another.

The distributed ACES system would require communication between multiple processors as well as the graphical user interface (GUI). This requires a great deal of control by

the processors with a variety of computations and communications being necessary at different times throughout the scenario. With the goal being to distribute the system such that each regionalized area can be processed as quickly as possible with transparency between regions a load balancing algorithm must try to ensure that the processors are utilized to their fullest throughout the engines execution.

3 HPC Load Balancing

The goal of processor allocation is to create the mapping in such a way so as to get better performance and better utilization of the processors available than would have been possible otherwise[4]. This problem is an NP-complete optimization problem [15], yet is one of such practical application that numerous researchers have examined the effectiveness of a variety of heuristics for solving the problem including neural networks [15], Recursive Spectral Bisection [19], diffusion method[12], [6], Evolutionary Strategies [14], and a variety of others way too numerous to cite.

For this analysis we define specific objectives so that the problem may be represented as a minimization problem with an assumed homogenous HPC cluster running p processors and n processes. For this algorithm we have chosen one logical constraint based on the number of processors allocated to each process and four innovative objectives. The four objectives selected are: request cost ($C_{request}$), preemption cost ($C_{preempt}$), communication linkage cost (C_{link}), and rollover cost ($C_{rollover}$)[4].

3.1 Constraints

Since some processes require upper and lower limits on the number of processors that they can support we must define the linear constraints binding them to these limits. If we allow $Required(i)$ and $Requested(i)$ to define the number of processors that process i must have to execute and the maximum allowed to execute respectively. Then to bound the number of processors that can be used we must ensure that no more than $Requested(i)$ are used, and that no less than $Required(i)$ are used, i.e.

$$Required(i) \leq \mid S_n(N_i) \mid \leq Requested(i) \, \forall i \in \{1 \dots n\} \tag{1}$$

3.2 Request Cost

The first objective that is to be dealt with is the request cost, $C_{request}$. Each process has some upper and lower limit on the number of processors that it can support and that it needs to operate respectively. These upper and lower limits create bounded constraints on the number of processors that can be allocated to each process. As shown in Equation 1 this constraint does not allow for any advantage for the process for using any more than the lower bound of processors. Thus the request cost objective exists in order to try to provide some benefit for using more than the lower bound of processors. For any given process the number of extra processors used is found using Equation 2.

$$C_{request_i} = \mid S_n(N_i) \mid - Required(i) \tag{2}$$

3.3 Preemption Cost

For the cost of preemption, $C_{preempt}$ we want to minimize the overall cost of preempting a process. There are two manners of preemption examined: the true preemption, that which occurs when a process is stopped so another process can use that processor, and the preemption that occurs when a process is moved to a different processor. Since not all processes can be preempted the feasibility of a solution based on it's being preemptive or not is dealt with as a feasibility constraint. For those processes that can be preempted we decided to examine the two types of preemption from the perspective of new processors added to a preexisting processor, and old processors being removed from a preexisting process. In this manner we can examine not only the preempted processors, but also include in it any additional cost that occurs from adding extra processors to an already ongoing process.

For the first type of preemption, where processes are added to a preexisting processor, we use the cost function as given by Equation 3.

$$C_{preempt+_i} = \mid S_n(N_i) - S_o(N_i) \mid \tag{3}$$

This equation enumerates, for some preemptible process i, the number of processors that exists in the potential next-state of the system that did not exist in the previous-state of the system.

Using the same logic, Equation 4 calculates the number of processors that existed in the old-state of the system that are not present in the potential next-state of the system for process i. In other words they determine the number of processors given to and removed from task i respectively.

$$C_{preempt-_i} = \mid S_o(N_i) - S_n(N_i) \mid \tag{4}$$

When combining these two objectives we have to understand that they are opposing objectives. In other words we want to maximize $C_{preempt+}$ and minimize $C_{preempt-}$. Also, we must ensure that the objective value of adding new processors to the current state ($C_{preempt+}$) is not nearly good enough to cover removing processors from the original state ($C_{preempt-}$). This is because of the overhead cost that either adding or removing processors to a currently executing process would incur. By adding new processors we ensure that executing program have to send out the appropriate commands, communicate whatever variables might be present, as well as potentially having to redistribute itself. While this is all detrimental to the run time, overall the addition of an extra processor should be able to benefit the executing task (assuming of course appropriate distributed system utilization in the process). On the other hand $C_{preempt-}$ forces the processor to do all the redistribution of load, but does not have the benefit of allowing for extra processor utilization. Thus $C_{preempt-}$ must be weighted more than $C_{preempt+}$. We arbitrarily chose a ratio of $\frac{1}{10}$ for a weight to distinguish between the two, giving us the equation:

$$C_{preempt} = C_{preempt-} - \frac{1}{10} C_{preempt+} \tag{5}$$

3.4 Link Cost

The link cost, C_{link} is a measure of the overall communication overhead experienced by each process. By using a normalized communication matrix, calculated a priori to the GA execution, we are able to take into account all communication costs that would be associated due to the relative node speeds, the backplane structure, etc. Thus for each process:

$$C_{link_i} = \sum_{j=1}^{(|S_n(N_i)|-1)} \sum_{k=j}^{(|S_n(N_i)|)} l(S_n(N_i))_{jk} \quad (6)$$

as the summation of all possible communication occurrences that could occur for process i.

3.5 Rollover Cost

The final cost chosen for examination is that of the rollover cost $C_{rollover}$. This is the cost associated with not including available processes for execution in the next-state of the system. The number of rolled-over processes is found from Equation 7. This is a critical element of the algorithm in that we want to ensure that as few processes are not executed as can be contained within the constraints as explained in section 3.1.

$$C_{rollover} = m - n \quad (7)$$

3.6 Normalization

Now that we have identified the objectives that are incorporated into the load balancing system we must adapt them so that they can be reasonably compared with each other. As can be seen by Table 1 the domains are too diversified to be able to be used directly in the objective function. Also, most of the objectives only take into account a single process and its relative configuration. Thus in order to find a unifying objective function we expand the cost functions to be able to incorporate the entire system in a normalized manner. The normalization approach taken for each of the objectives is based off of the standard statistical normalization equation. The normalized multiobjective symbolic metrics are given in Table 1.

4 MOPs and MOEA Design

A multiobjective optimization problem (MOP) consists of decision variables, two or more objective functions, and constraints. Standard MOP and MOEA definitions and nomenclature can be found in [8]. Such symbolic formulation includes feasible regions in objective space, feasible solutions, solution dominance and non-dominance, true and approximate Pareto optimal solutions and Pareto Front (P*/PF*), current computational values (P_{known}/PF_{known}), fitness sharing, niche count, sharing function, mating restrictions, ranking and the required evolutionary algorithm characteristics. The goal of a Pareto-based MOEA is convergence of PF_{known} towards PF*. MOEAs generally operate on a population of candidate solutions (chromosomes) as opposed to a single solution; therefore, the strength of a MOEA is its ability to find multiple non-dominated solutions (P_{known}) that are close to P*.

Evolutionary algorithms (EA) consist of a class of algorithms that use the concepts of genetics to enable them to explore the search space landscape for a specific optimization problem domain. The class includes genetic algorithms (GA), evolution strategies (ES), and evolutionary programming (EP). EA's have been used for a wide variety of stochastic search applications, including processor allocation problems [14] [9] [3]. In an evolutionary algorithm, a collection or group of individuals is initially defined; each individual is known as chromosomes; and a group is defined as a population. The population moves through generations by using genetic operators such as mutation and recombination. Mutation works by inserting new genetic material into the population by modifying individuals, recombination by transferring preexisting genetic material between two or more individuals of the population. There are many varieties of genetic operators each with a different set of operational parameters that may be modified [1].

For our MOEA development, the GENOCOP III GA software was selected as a foundation because of its generic breath of application. The GENOCOP III algorithm is a well known GA package that uses a variety of arithmetic operators for optimizing real-valued alleles [17]. In the III version of this GA software, a collection of constraint handling operations have also been implemented[18]. We have extended this GENOCOP software by integrating a multiobjective Pareto optimization capability [5]. Thus, the extended GENOCOP III is a general multiobjective optimization algorithm and is labelled as **MOCOP**.

Note that GENOCOP III is a real-valued EA and the resulting choices in our applications are supposed to be integer solutions. For the obvious reason that we would not desire a fractional computing system for the specified applications, we truncate all of the chromosome elements (alleles) so that they can easily represent any computer system set of processors. This is not a problem because of the fact that we constrained the domain of the alleles to be between 0 and $n + 1$ in the implementation; therefore, our possible genotype solutions are appropriate. Other integer-based software was considered, but the breath of GENOCOP III and ease of implementation indicated to the authors that it was the better choice.

There exists multiple implementations for storing the secondary population of $P_{known}(t)$ as originally defined in the GENOCOP real-valued genetic algorithm. One common approach is to simply add the current population at each step (ie- $P_{known}(t) := P_{current}(t) \bigcup P_{known}(t-1)$ and periodically removing any dominated chromosomes from the population. This removal process is done through the use of what is termed Pareto Ranking. In general, the size of the non-dominated population or set is a concern in a given application. In order to restrict the population size within memory limitations, a number of techniques can be applied. For example, in order to attempt a uniform distribution of solutions across the Pareto Front, an operator to ensure Fitness Sharing can be used. Many varieties of the

Standard Function	Normalized Function
$C_{request_i} = \mid S_n(N_i) \mid - Required(i)$	$C_{request} = \frac{\sum_{i=1}^{n_n}(\mid S_n(N_i)\mid - Required(i))}{\sum_{i=1}^{n_n}\mid S_n(N_i)\mid}$
$C_{preempt+_i} = \mid S_n(N_i) - S_o(N_i) \mid$	$C_{preempt} = \frac{\sum_{i=1}^{n_o}\mid S_n(N_i) - S_o(N_i)}{1+\sum_{i=1}^{n_o}\mid S_n(N_i)\mid}$
$C_{preempt-_i} = \mid S_o(N_i) - S_n(N_i) \mid$	$C_{preempt-} = \frac{\sum_{i=1}^{n_o}\mid S_o(N_i) - S_n(N_i)}{1+\sum_{i=1}^{n_o}\mid S_n(N_i)\mid}$
$C_{link_i} = \sum_{j=1}^{\mid S_n(N_i)\mid -1}\sum_{k=j}^{\mid S_n(N_i)\mid} l(S_n(N_i))_{jk}$	$C_{link} = 1 - \left(\frac{\sum_{i=1}^{n_n}\left(\sum_{j=1}^{(\mid S_n(N_i)\mid -1)}\sum_{k=j}^{(\mid S_n(N_i)\mid)} l(S_n(N_i))_{jk}\right)}{\sum_{j=1}^{p-1}\sum_{k=j}^{P} l(S_n)_{jk}}\right)$
$C_{rollover} = m - n$	$C_{rollover} = \frac{m-n}{m}$

Table 1: Standardized Objective Functions for the Allocation EA

ranking and fitness sharing operators exist and an examination of these operators can be found in [20].

Once the Pareto Front has been found, the job of the DM is to select which point would be best for the needs of the specific project. Thus making Pareto Optimization an *a posteriori* technique. With proper operators this formulation can find fully enumerated optimal surfaces for which dynamic selection at run time is available, depending on the problem, of equally non-dominated solutions.

A Pareto ranking scheme was incorporated into MOCOP using Fonseca and Fleming's Ranking function:

$$rank(\vec{x}, t) = r_u^{(t)} \tag{8}$$

where $r_u^{(t)}$ is the amount of the population at generation t that dominates x_u. In this manner the fitness value of each chromosome are measured according to its Pareto rank as opposed to the weighted sum of the aggregate approach.

The original GENOCOP III chromosome consisted of a vector $\vec{x}$ where $\vec{x}(0) = f(\vec{x})$. Because Pareto fitness is based off of not one but multiple objective values this is modified such that

$$\vec{x}(0), t = rank(\vec{x}, t) \tag{9}$$

where the ranking is based on the fitness values returned by each of the objectives. Once again the values are mapped to the interval $[0, 1]$.

MOCOP functionality provides for the ranking of the entire population added as well as a member comparison function. This function takes two members of the population and ranks them against the whole population comparing their resulting rank and returning which rank is greater. The populations are expanded to store the objectives, thus all routines that copy the population members are modified as well to incorporate the objective values.

Regarding our load balancing application, the chromosome representation implemented assigns each element of the chromosome as a processor on the high performance computing system. Thus the value at the allele represents the process ID number that is assigned to the processor that is mapped to that allele.

5 Design of Experiments

For analysis, the experiment is executed on two different problems that reflect essential characteristics of the two applications. Both problems are based on a 16 processor system, with four processes trying to execute and an assumed mesh backplane where process communication costs between processors is directly proportional to their distances. In the first problem (Problem 1) none of the processors are initially allocated. The second problem (Problem 2) has all of the processors initially allocated uniformly except for one empty processor.

Each experiment is run ten times on a Pentium 4 1.7Ghz system using the initialization parameters as given in Table 2. Given this data, computational results are statistically calculated using the metrics of Overall Non-dominated Vector Generation(ONVG) and Spacing. Results are depicted in Table 3. These metrics were selected because of the unknown PF* and thus, relative metrics were desired. Selection of other possible metrics is possible depending upon the desired evaluation criteria [8].

6 Comparative Pareto Fronts

An example of the phenotypic results of a solution of non-dominated vectors is given in Figure 1. Figure 2 depicts the specific tradeoffs between combinations of objectives. The rollover cost are embedded into each of the charts. Each shape represents the number of processes that were not included as part of the generated solution.

In order to select a load balancing configuration for a distributed system an examination of the tradeoffs between each of the objectives is necessary. This examination is done based on the priorities of the intended application suite to be executed on the HPC system. Regardless of the intended application, if a process is to be executed it should be started as soon as possible. For this reason the first objective that is analyzed is the rollover cost. This allows the decision maker to ensure that the system avoids rollovers whenever possible. However, this is not to suggest that an aggregated single objective method be employed instead. As can be seen from Figure 2, chart C, there is clear relationship between the number of requests able to be satisfied and the rollover cost. This is also rationally correct, for as less pro-

Parameter	Setting
Reference Population Size	50
Search Population Size	50
Total Number of Evaluations	10000
Reference Population Evolution Period	20
Number of Offspring for Ref. Population	50
Search Point Replacement Ratio	0.4
Reference Point Initialization Method	1 (Multiple)
Search Point Initialization Method	1 (Multiple)
Frequency of Operators	10 (For All)

Table 2: GENOCOP III Parameters From Generic Experiments

	Mean ONVG	Variance ONVG	Mean Spacing	Variance Spacing	Mean Time	Variance Time
Problem 1	34.8	3.7	0.379	0.0022	80.75s	1.5833
Problem 2	11.6	42.8	0.2404	0.0036	79.2	2.7

Table 3: Multiobjective Results for the Allocation Problems on a 16 Processor, 4 Process System

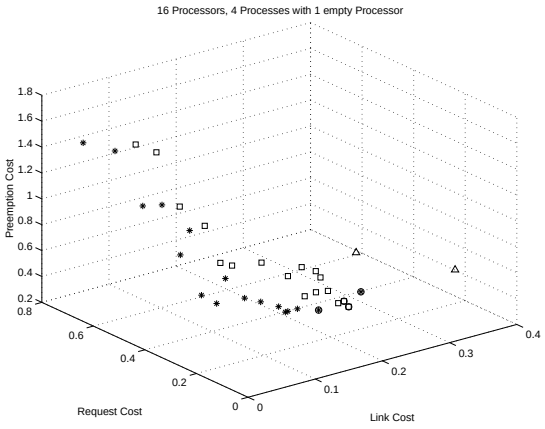

Figure 1: PF_{known} for a 16 Processor, 4 Process load balancing problem. The stars represent allocations that allocated all processes to processors, squares allocate all but one, and triangles allocated all but two processes. The circles represent those solutions that utilize every processor.

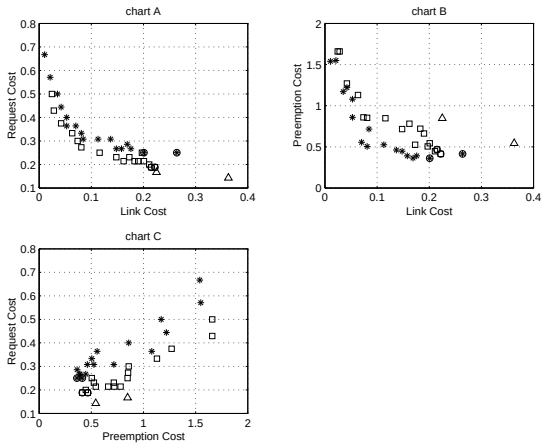

Figure 2: Comparison of specific objective tradeoffs.

cesses are executed more processors are available for the remaining processes to utilize.

Comparing the cost of communication to the number of processes not included, it can be seen that as less processes are included the higher the communication cost. Since fewer processes means more processors per process the communication being higher between these processors is understandable.

When examining the request costs with respect to the other objectives it appears that the request cost is inversely proportional to link cost (Figure 2,chart A) and directly proportional to preemption cost (Figure 2,chart C). As the number of processors allocated to each process is increased $C_{request}$ lowered (see Equation 2), but with the increase in processors per process an increase in communication overhead occurs and hence C_{link} increases as shown by Equation 6. Due to the stochastic nature of the MOEA, the allocation that is input to the system as the initial state does not imply that the result is the same, unless the processes specifically state that they are not preemptible. Since for this experiment all processes are assumed to be preemptible, processes are allowed to roam from their initial configuration. From Figure 2, chart C, we see that most of the requests and preemptions lie in lower region of the objectives. Those solutions that are in the higher area of the objective space do so for the communication cost, trading preemption and request costs for lower link costs, as shown in Figure 1 and Figure 2, chart B.

Another important factor that should be examined that is not reflected in the objectives is the number of unutilized processors in the system. The addition of extra processors to a process is not always beneficial, and thus should not be forced upon applications [16]. As such the addition of an extra objective for minimizing the number of unused processors was not included. The average number of unused processors for a Pareto Front with 38 solutions is 3.94 ± 0.318 with a min of 0 and a max of 10. For the experiment presented in Figures 1 and 2, the solutions marked

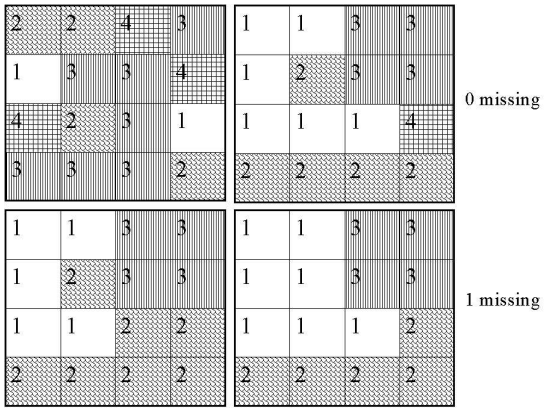

Figure 3: Genotypic results (Pknown) of the fully utilized processors, the top row shows the results that had all four processes being included, the bottom has three processes being included. Each block of the graph represents a processor with the number and shading representing a process assigned to that processor.

Priority	ACES	CHSSI
1	Rollover Cost	Rollover Cost
2	Request Cost	Preemption Cost
3	Preemption Cost	Link Cost
4	Link Cost	Request Cost

Table 4: Objective Prioritization Based on Application

with a circle represent those that utilize all available processors. As can be seen, for this experiment, there are four solutions that utilize all available processors. Two of these solutions allocate all of the processes, two allocate all but one. All four points have a relatively higher request cost, which is again explainable based on the fact that there are less processors available since they are all being utilized. The genotypic results are shown in Figure 3 where it is evident that the processes are beginning to cluster into groups for minimizing communications.

6.1 Application Specific Regions

In order to apply a load balancing solution to a specific problem the Pareto front must be examined with regards to the objective tradeoffs deemed important for the application. As shown in Table 4 each of the applications discussed in Section II have different prioritization based on the requirements of the application. This prioritization defines the region of the Pareto Front that should be examined to best benefit the application. Also, observe that the non-dominated population size was quite small in the two example discrete problems in part because of the constraints.

For the discrete event simulation algorithm, every process must be included with every set of calculations($C_{rollover}$). This is the primary goal because for any missed time frame of calculations we would upset the balance created by the overall system. The next most important objective would be to allow requested processors to execute as provided by $C_{request}$.

DES systems in general have a variable number of computations that must be executed throughout the computation time. By allowing additional processors to support the available applications, as emphasized by the $C_{request}$ objective, we are able to diffuse the processing done by the system as a whole. This helps to ensure a quick runtime for next state of the DES system.

As with any distributed system the communication overhead and preemption policies are important, for a DES system they are not as important as the other two objectives. The preemption of processes (assuming it is allowed for the given application with given process priorities) would allow for different parts of the system taking and losing processors as needed and thus help to increase the number of processors given to requesting processes at different time steps. As for C_{link}, this is the least important of the four since by the very nature of the Pareto front the communication choices provided are those that dominate other communication layouts of the search space.

When examining the Pareto front with respect to Real Time Digital Signal Processing we again want to ensure that we are executing every process available so that the real-time data can be quickly analyzed. As opposed to the DES system which has rapidly changing processor requirements, the input from the signal collectors produces data at a steady rate. Thus $C_{request}$ would have the lowest priority of the four objectives to be examined. For this application $C_{preempt}$ would be one of the more prominent objectives. Since the processes act in a predictable fashion based on a steady amount of data that they receive, and the fact that this data is being received in a real-time fashion, it would be cost prohibitive to preempt the processes. Thus, minimizing preemption should be high priority. This leaves communication link cost (C_{link}) as the third most important objective. This is more important than the $C_{request}$ objective due to the large number of intra-processor communication that must be done in order to do the mathematical calculations that are necessary.

7 Conclusions

We have shown the relative tradeoffs of four load balancing objectives with respect to the general characteristics of two real-world applications ACES discrete event simulation and CHSSI/SIP DSP analysis. Further, the MOCOP algorithm, as an extension of the GENOCOP III EA software, can provide an effective Pareto front mapping for the load balancing of these or possibly any generic or specific application. Specific engineering and architectural parameter values for distributed processor allocation would be input to MOCOP. The specific region of the Pareto front must be selected by the decision maker only after an examination of the tradeoffs of the objectives. Then using a prioritization based on the requirements of the application an effective solution region can be determined and implemented.

Whether using an *a priori* aggregate approach or an *a posteriori* Pareto approach, analysis of the objective tradeoffs is important. By examining the objectives the decision

maker is able to select which region of the solution space is most effective for the intended application. If the region is stable then the decision maker might choose to use an aggregate approach so that the selection can be automated. By examining the solution region using the *a posteriori* approach the decision maker can more carefully tune the weights so that they reflect the solution region that is most beneficial for the application. While this requires more up-front analysis it allows for the system to be more automated when deployed to an actual working environment.

Future investigation includes extension to much large dimensional problem domains, development of additional objectives and constraints for the processor scheduling problem, study and analysis of non-dominated population size restrictions, and comparison to other integer-MOEA performance.

8 Acknowledgements

The authors would like to thank Dr. Richard Linderman at the Air Force Laboratory, Information Directorate (AFRL/IFTC - HPC CHSSI SIP project) and Mr. Jon DiLeo at the Air Force Wargaming Institute (AFWI/WGT) for their support of this research.

Bibliography

[1] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[2] Jerry Banks, II John S. Carson, and Barry L. Nelson. *Discrete-Event System Simulation*. Prentice-Hall, 1995.

[3] Christopher A. Bohn and Gary B. Lamont. Load balancing for heterogeneous clusters of pcs. *Future Generation Computer Systems*, 1(18):389–400, 2002.

[4] David J Caswell. Active processor scheduling using evolutionary algorithms. Master's thesis, Department of Electrical and Computer Engineering. Air Force Institute of Technology, WPAFB, Ohio, Dec. 2002.

[5] David J. Caswell and Gary B. Lamont. Wire-antenna geometry design with multiobjective genetic algorithms. In *Proceedings of the 2001 International Conference on Evolutionary Computation*, Piscataway, New Jersey, 2001. IEEE.

[6] Y. Chan, S. Dandamudi, and S. Majumdar. Performance comparison of processor scheduling strategies in a distributed-memory multicomputer system. In *Proc. Int. Parallel Processing Symp (IPPS)*, pages 139–145, 1997.

[7] A. Choudhary, W.K. Liao, P. Varhney, D. Weiner, R. Linderman, and M. Linderman. Design, implementation and evaluation of parallel pipelined stap on parallel computers. In *12th International Parallel Processing Symposium*, 1998.

[8] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002.

[9] Diane Cook, Joey Baumgartner, and B. Shirazi. Genetic solutions to the load balancing problem in parallel computers. In *Proceedings of the 1995 International Conference on Parallel Processing*, 1995.

[10] Ingemar J. Cox and Matt L. Miller. On finding ranked assignments with application to multitarget tracking and motion correspondence. *Trans. Aerospace and Electronic Systems*, 31:486–489, 1995.

[11] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling block-cyclic array redistributions. In *IEEE Transactions on Parallel and Distributed Systems*, volume 9. IEEE, February 1998.

[12] Hluch'y Dobrovodsk'y Dobruck'y. Static mapping methods for processor networks. In *Centre for Parallel Computing, University of Westminster, London*, 1997.

[13] Douglas F. Elliot and Ramamohan K. Rao. *Fast Transforms Algorithms, Analyses, Applications*. Academic Press, Inc., New York, 1982.

[14] G. Greenwood, A. Gupta, and K. McSweeney. Scheduling tasks in multiprocessor systems using evolutionary strategies. In *Proc. 1st IEEE Conf. on Evolutionary Computation*, pages 345–349, 1994.

[15] Tracy Braun Howard, Jay Siegel, and Noah Beck. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, pages 810–837, 2001.

[16] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing Design and Analysis of Algorithms*. The Benjamin/Cummings Publishing Company, 1994.

[17] Z. Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 2:647–651, 1995.

[18] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.

[19] Horst D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.

[20] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147, 2000.