An Artificial Immune System for Heterogeneous Multiprocessor Scheduling with Task Duplication

Young Choon Lee and Albert Y. Zomaya
Advanced Networks Research Group, School of Information Technologies,
The University of Sydney, NSW 2006, Australia
{yclee,zomaya}@it.usyd.edu.au

Abstract

In this study, we investigate the task scheduling problem in heterogeneous computing environments and propose a novel scheduling algorithm, called the Artificial Immune System with Duplication (AISD) algorithm that efficiently tackles the problem. The AISD algorithm incorporates the clonal selection principle in the immune system and task duplication into the scheduling process. Based on the performance results obtained from extensive experiments conducted with a comprehensive set of both randomly generated and well-known application task graphs and various system configurations, AISD consistently outperformed the two existing algorithms by a noticeable margin, especially when scheduling communication intensive task graphs.

1. Introduction

Task scheduling problems have been extensively studied for many years. However, due to the NP-complete nature of the problem in most cases heuristic algorithms account for a myriad of existing scheduling algorithms [1]. Most of these scheduling heuristics have been designed for homogeneous computing systems, whereas only a handful of efficient heuristics for heterogeneous computing systems has been proposed. Heuristic based scheduling algorithms are normally the ones favored by a large number of researchers.

In attempts to obtain better schedules a noticeable number of metaheuristics, mostly inspired by nature, including genetic algorithms, ant colonies, tabu search and simulated annealing, have also been adopted. More recently, another biologically-inspired approach emerged which is based on the immune system.

The immune system of an organism, the vertebrate immune system in particular, is regarded as an extremely effective defence system against virtually an unlimited number of biological attackers, such as microorganisms, parasites and viruses. It protects the host from these invaders by coordinating a complex, yet sophisticated set of immune features. Due to the self-organizing, cooperative and robust characteristics of the immune system its applicability to various areas, such as pattern recognition, network security and anomaly detection has been actively studied in the past decade [2, 3, 4]. However, only recently immune system based scheduling algorithms have been developed by researchers to target different instances of the problem [5, 6, 7, 8].

This paper proposes an artificial immune system for heterogeneous multiprocessor scheduling with task duplication known as the Artificial Immune System with Duplication (AISD). It first generates and refines a set of schedules using a modified clonal selection algorithm and then attempts to further improve the schedules with task duplication. The AISD algorithm schedules tasks in a task graph via three carefully designed phases: clonal selection, task duplication and ineffectual task removal. The performance gain of the proposed algorithm is obtained not from a particular one of these phases, but from the careful coordination of them. Despite the adoption of immune features (i.e., clonal selection and affinity maturation), task insertion and task duplication, the AISD algorithm remains at an affordable level of time complexity.

The remainder of this paper is organized as follows. Sections 2 and 3 introduce some background material on scheduling problems and the immune system, respectively. The proposed algorithm is described in detail in Section 4. In Section 5, the evaluation results are presented and explained with conclusions following in Section 6.

2. Scheduling Problem

Parallel programs, in general, can be represented by a directed acyclic graph (DAG). A DAG, $G = (V, E)$, consists of a set V of v nodes and a set E of e edges. A DAG is also called a task graph or macro-dataflow graph.

1-4244-0910-1/07/$20.00 ©2007 IEEE

In general, the nodes represent tasks partitioned from an application and the edges represent precedence constraints. An edge $(i, j) \in E$ between task n_i and task n_j also represents the inter-task communication. In other words, the output of task n_i has to be transmitted to task n_j in order for task n_j to start its execution. A task with no predecessors is called an *entry* task, n_{entry}, whereas an *exit* task, n_{exit}, is one that does not have any successors. Among the predecessors of a task n_i, the predecessor which completes the communication at the latest time is called the most influential parent (MIP) of the task denoted as $MIP(n_i)$.

The weight on a task n_i denoted as w_i represents the computation cost of the task. In addition, the computation cost of the task is on a processor p_j, is denoted as $w_{i,j}$ and its average computation cost is denoted as $\overline{w_i}$.

The weight on an edge, denoted as $c_{i,j}$ represents the communication cost between two tasks, n_i and n_j. However, communication cost is only required when two tasks are assigned to different processors. In other words, the communication cost when they are assigned to the same processor can be ignored, i.e., 0.

The target system used in this work consists of a set P of p heterogeneous processors/machines that are fully interconnected. The inter-processor communications are assumed to perform with the same speed on all links without contentions. It is also assumed that a message can be transmitted from one processor to another while a task is being executed on the recipient processor which is possible in many systems.

The earliest start time of, and the earliest finish time of, a task n_i on a processor p_j is defined as

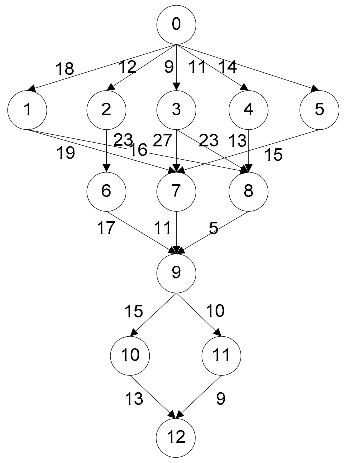
$$EST(n_i, p_j) = \begin{cases} 0 & \text{if } n_i = n_{entry} \\ EFT(MIP(n_i), p_k), p_k \in P & \text{if } j = k \\ EFT(MIP(n_i), p_k), p_k \in P + c_{MIP(n_i),i} & \text{if } j \neq k \end{cases}$$

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j}$$

Note that the actual start and finish times of a task n_i on a processor p_j, denoted as $AST(n_i, p_j)$ and $AFT(n_i, p_j)$ can be different from its earliest start and finish times, $EST(n_i, p_j)$ and $EFT(n_i, p_j)$, if the actual finish time of another task scheduled on the same processor is later than $EST(n_i, p_j)$.

In the case of adopting task insertion the task can be scheduled in the idle time slot between two consecutive tasks already assigned to the processor as long as no violation of precedence constraints is made. This insertion scheme would contribute in particular to increasing the processor utilization for a communication intensive task graph with fine-grain tasks.

A simple task graph with its details is shown in Figure 1. The values presented in the last column of the table in Figure 1 are computed using a frequently used task prioritization method, *b-level*. The b-level of a task is



Task	P0	P1	P2	b-level
0	14	16	9	162.33
1	13	19	18	129.67
2	11	13	19	137.33
3	13	8	17	132.67
4	12	13	10	121.67
5	13	16	9	120.67
6	7	15	11	100.00
7	5	11	14	93.00
8	28	12	20	97.00
9	14	7	16	72.00
10	16	25	12	44.67
11	17	10	20	38.67
12	9	14	19	14.00

Figure 1. A sample task graph

computed by adding the computation and communication costs along the longest path of the task from the exit task in the task graph (including the task).

The communication to computation ratio (CCR) is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system.

The task scheduling problem in this study is the process of allocating a set V of v tasks to a set P of p processors aiming to minimize schedule length (SL), also called *makespan*, without violating precedence constraints. The schedule length is defined as $SL = max\{AFT(n_{exit})\}$ after the scheduling of v tasks in a task graph G is completed.

3. The Immune System: Principles and Processes

The immune system is a biological defence mechanism designed to protect a given organism primarily from microbes, such as bacteria, archaea, fungi, protists and viruses. Using allied forces of cells, tissues and organs, it battles against foreign invaders.

At the highest level, two defence lines (the innate and the adaptive immune systems) are embodied. The core forces of both systems are different types of white blood cells.

The innate or non-specific immune system is the first line of defence that uniformly combats any invader very directly and immediately with chemical substances and specific types of white blood cells. However, during the lifetime of an organism it encounters numerous different attackers (antigens) that the innate immune system is not able to handle effectively. The adaptive or specific immune system comes into play in such a circumstance.

Among a number of immune features in the adaptive immune system clonal selection with affinity maturation is the particular interest in this study in that it is incorporated with the proposed algorithm.

3.1. Adaptive Immune System

As a host experiences constant encounters of various antigens it is quite necessary for immune entities to be equipped with memory, learning and adaptive functions. The adaptive immune system protects the host by a sophisticated coordination of these functions.

The two key components in the adaptive immune system are B lymphocytes (B cells) and T lymphocytes (T cells) of white blood cells that are produced by stem cells in the bone marrow. While T cells take charge of the cellular immunity B cells, more precisely immunoglobulins or antibodies, oversee the humoral immunity.

Now the question is how the adaptive immune system can respond against a virtually unlimited and diverse set of antigens. A sequence of phases for battling against these immunological enemies shown in Figure 2 can answer this question.

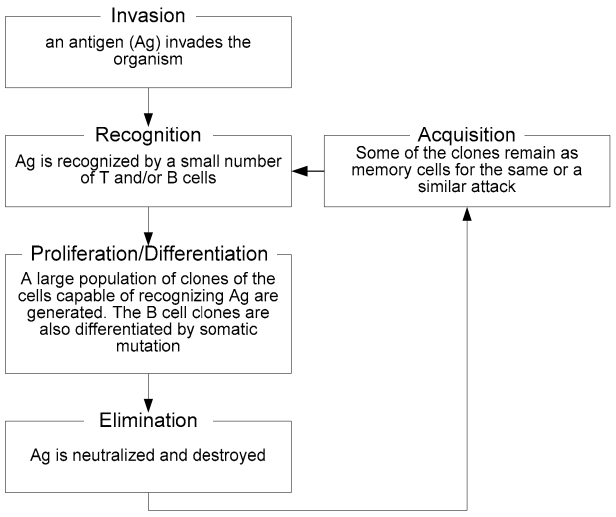

Invasion
an antigen (Ag) invades the organism
Recognition
Ag is recognized by a small number of T and/or B cells
Acquisition
Some of the clones remain as memory cells for the same or a similar attack
Proliferation/Differentiation
A large population of clones of the cells capable of recognizing Ag are generated. The B cell clones are also differentiated by somatic mutation
Elimination
Ag is neutralized and destroyed

Figure 2. The primary steps involved in the adaptive immune system

3.2. Clonal Selection

One of the most powerful features of the immune system is its adaptability. The clonal selection principle [9] in the adaptive immune system plays an important role in this property. Although clonal selection occurs on both T and B cells the focus in the field of AIS is often aimed at B cells. This is primarily due to the fact that B cell clonal selection involves mutation that further enhances the adaptability of B cells. Hereafter, clonal selection simply refers to that of B cells.

The rationale behind the clonal selection theory is that superior B cells are preserved with a minor degree of mutation and become prevalent, and inferior ones are mutated at a high rate hoping to be improved and become rare. More specifically, when a foreign intruder (antigen) attacks the host, B cells matching the antigen will be cloned (i.e., clonal expansion) and mutated (i.e., affinity maturation) at rates directly proportional to and inversely proportional to the degree of the match (or affinity), respectively. Note that the superiority of a B cell actually refers to that of its antibody.

3.3. Artificial Immune Systems

The biological immune system has been shown to be a great mechanism to effectively deal with a virtually unlimited number of threats in very stochastic environments. It has been used in an increasing number of areas, such as computer and network security, data mining and pattern recognition. An artificial immune system (AIS) can be defined as a methodology for solving a real-world problem by using abstractions inspired by features of the immune system. Several other definitions can be found in the literature [10].

The two most favored immune entities modeled in AIS are antibodies and antigens since they are key players in the adaptive immune system. In cooperation with these immune entities several other immunological theories, such as negative/positive selection, clonal selection, immune networks and danger model have been actively modeled [11]. Note that, these are only some of the popular instances that have been modeled using a rich set of immune characteristics.

4. The Artificial Immune System with Duplication (AISD) Algorithm

The AISD algorithm performs scheduling through three major phases, a clonal selection phase, a task duplication phase and an ineffectual task removal phase in this order. The workings of these three phases are shown in Figures 3, 4 and 5, respectively.

4.1. Clonal Selection Phase

The AISD algorithm adopts the clonal selection principle in the immune system as a key player to generate and refine schedules, eventually resulting in good quality schedules. Here, a schedule is denoted as an antibody. The heuristic achieves its competitive performance not by simply incorporating immune features, but by intuitively

```
Function ClonalSelection
/**
Input:
  minG: min #groups;
  maxG: max #groups;
  c: #clones to generate for each base antibody
  sr: maximum antibody selection rate
  basemr: mutation rate
Output:
  A set AB of antibodies                                  **/
1. Compute $b\text{-}level(n_i)$ $\forall n_i \in V$
2. Sort the tasks $\in V$ in decreasing order by b-level value
3. Let $b = v / (minG + v / p \% maxG)$
4. Let $AB = \emptyset$
5. while ( $\exists n_i \in V$ | $n_i$ is unscheduled) do
6.     Remove first $b$ tasks from $V$ and insert them to $B$
7.     Let $newAB = \emptyset$
8.     do
9.         Let $prevab$ = the first antibody in $AB$
10.        Generate a base antibody $baseAB(B)$ based on
           $\min_{p_j \in P} AFT(n_i, p_j)$ taking $prevab$ into account $\forall n_i \in B$
11.        Let $abP = prevab + baseAB(B)$
12.        Generate a set $C$ of $c$-1 clones of $baseAB(B)$
13.        for each clone $c_k \in C$ do
14.            Mutate $c_k$ based on $basemr$ /* $b \times basemr$ times */
15.            Add ($prevab + c_k$) to $abP$
16.        end for
17.        Let $AB = AB - prevab$
18.        Add $abP$ to $newAB$
19.    while ($AB \neq \emptyset$)
20.    Select best antibodies in $newAB$ based on $sr$ and store
       them in $nextAB$
21.    for each $ab_i \in nextAB$ do
22.        Let $c_i$ = a clone of $ab_i$
23.        Mutate $ab_i$ based on affinity (schedule length)
24.        Replace $ab_i$ with $c_i$ if $sl(c_i)$ shorter than $sl(ab_i)$
25.    end for
26.    Let $AB = nextAB$
27. end while
28. return AB
```

Figure 3. The Clonal selection algorithm

adapting and carefully coordinating them for scheduling purposes.

The adaptations to typical characteristics of clonal selection AISD makes include group-wise task scheduling and judicious base schedule generation. That is, tasks in a given task graph are grouped into a number of scheduling batches based primarily on the number of processors with time complexity taken into consideration (step 3). Tasks in each batch are initially scheduled based on their earliest actual finish times (step 10). The schedule of the batch is called a base antibody (schedule). This base antibody should not violate any precedence constraints; that is, the schedule for tasks in the previous batches is considered when computing the earliest actual finish times of the tasks in the batch. This preceding schedule is denoted as *prevab* at step 9. The base antibody is cloned and the clones are mutated based on a uniform mutation rate specified as an input parameter (steps 13-16). This mutation process in addition to the base antibody generation scheme is adopted in order to ensure the antibody population is generated at a certain level of quality. Each antibody in an antibody population is a concatenation of the preceding antibody and a currently generated one (step 15). After generating a set of antibody populations antibodies in each population are evaluated based on schedule length and a set of best ones are selected, with the probability of selection directly proportional to affinity. More formally,

$$NAB(abP) = max\{0, |abP| \times (sr - (ssl(abP) - mssl) / mssl)\},$$

where *NAB(abP)* is the number of antibodies to select in antibody population *abP*, *sr* is the maximum antibody selection rate (e.g., 0.3 for 30%) specified as an input parameter, *ssl(abP)* is the shortest schedule length in *abP*, and *mssl* is the minimum shortest schedule length in all antibody populations. Note that, there is an inverse relationship between schedule length and affinity. These selected antibodies further undergo a process called hypermutation, with the probability of mutation inversely proportional to its affinity. The number of mutations per antibody is defined to be

$$NM(ab) = 2 + (sl(ab) - mssl) / mssl * 10 * \lambda,$$

where *NM(ab)* is the number of mutations antibody *ab* undergoes and *sl(ab)* is the schedule length of *ab*. The minimum number of mutations is set to 2. The number of mutations is characterized primarily by the mutation rate parameter λ.

Actual mutations are carried out by the two types of mutation (point mutation and point-point swapping) adopted in the proposed algorithm. Any time a mutation is required a mutation type is randomly selected. As the names of the two mutation types are self explanatory they perform random point by point replacements and point to point swaps, respectively. More specifically, for each mutation the former randomly selects a point (processor) in an antibody and replaces it with a randomly chosen one, whereas the latter swaps two randomly selected points. If the mutated clone of an antibody exhibits stronger affinity than that of the antibody, it supersedes the original antibody (step 24). This series of steps repeats until tasks in all batches are scheduled.

4.2. Task Duplication Phase

```
Function TaskDuplication
/**
Input: A set $AB$ of antibodies
Output: An antibody (schedule) of $V$ onto $P$        **/
1.  Let $sl(bestab) = \infty$
2.  for each $ab_i \in AB$ do
3.      for each task $n_{i,k}$ encoded in $ab_i$ do
4.          Compute $AFT(n_{i,k}, p_j)$ $\forall p_j \in P$
5.          Let $ndups = \min\{p-1, \max\{1, \#\text{ grand children of } n_{i,k}\}\}$
6.          Duplicate $n_{i,k}$ as many as $ndups$ times based on $AFT$
7.      end for
8.      Recompute schedule length of $ab_i$
9.      Replace $bestab$ with $ab_i$ if $sl(ab_i)$ is shorter than $sl(bestab)$
10. end for
11. return $bestab$
```

Figure 4. The task duplication algorithm

As a result of the clonal selection phase of AISD a set of best antibodies is obtained. Task duplication, as an attempt to further reduce the schedule length, then takes place with these resultant antibodies. Each task encoded in a selected antibody is considered for duplication. Tasks are duplicated only if duplications do not increase the schedule length associated with the selected antibody. The number of duplications ranges from at least one up to as many as whichever is the minimum: the number of its grand child tasks and one fewer than the number of processors (step 5). Note that there is at most one instance of a task on each processor. This duplication policy ensures that the higher priority and more successor tasks a task has the more duplications it is considered for. At the end of this task duplication phase the best antibody (the one with the shortest schedule length) is selected and passed into the ineffectual task removal phase.

4.3. Ineffectual Task Removal Phase

Now the best antibody, generated and selected via the clonal selection and task duplication phases, is scrutinized to see if there are any unnecessarily duplicated tasks. If so, those useless replicas are removed.

The first step in the ineffectual task removal phase is to ensure there is only one exit task scheduled. Since any task in a task graph can be duplicated at least once including the exit task there might be two copies of the exit task in the schedule (step 1).

For each task, its immediate predecessor (parent) tasks including their replicas are examined to find out which parent tasks are most effectively scheduled (steps 8-12). The decisions are made based on their actual finish times. For a particular task if it is not regarded to be useful for any of its child tasks after checking the usefulness of all its child tasks it is removed (step 6).

```
Function IneffectualTaskRemoval
/**
Input: An antibody $ab$
Output: A refined antibody $ab$                    **/
1.  Remove the one, among the replicas of the exit task
    in $ab$, whose actual finish time is the latest
2.  for each task $n_i$ from the back of $ab$ do
3.      Let $IP_i$ = a set of immediate predecessor tasks of $n_i$
4.      for each replica $r_{i,j}$ of $n_i$ do
5.          if ($r_{i,j} \neq n_{exit}$ and $useful(r_{i,j}) \neq$ true) then
6.              Remove $r_{i,j}$
7.          else
8.              for each immediate predecessor task $ip_{i,k} \in IP_i$ do
9.                  Let $RIP_{i,k}$ = a set of replicas of $ip_{i,k}$
10.                 Let $uip = \min_{rip_{i,k,m} \in RIP_{i,k}} \{AFT(rip_{i,k,m}, p_{i,k,m}) + c_{rip_{i,k,m}, r_{i,j}}\}$
11.                 Let $useful(uip)$ = true
12.             end for
13.         end if
14.     end for
15. end for
16. return $ab$
```

Figure 5. The ineffectual task removal algorithm

Note that the ineffectual task removal algorithm assumes that there is only one exit task in a task graph. In case of a task graph with multiple exit tasks a dummy exit task, to which the actual exit tasks are connected, is added. Thus, any costs (i.e., computation and communication) associated with this addition are set to 0.

5. Performance Evaluation

In this section the performance of the AISD algorithm is evaluated based on its performance results obtained from experiments conducted with two extensive sets of task graphs: randomly generated and well-known application task graphs. The three well-known parallel applications used for our experiments are the Laplace equation solver [12], the LU-decomposition [13] and Fast Fourier Transformation [14]. The proposed algorithm is also compared with two previously proposed heuristics, i.e., HEFT [15] and DBUS [16]. The selection was determined mainly by two main factors. The first is that they have both been shown to perform well in terms of the schedule length. Secondly, the target system configurations of the two heuristics are compatible with AISD. To the best of our knowledge, none of existing scheduling schemes for heterogeneous computing systems incorporates the immune system as a core component.

The comparison results in this work are presented with intermediate results of AISD, i.e., schedules generated by the AIS (the clonal selection phase) without the task duplication and ineffectual task removal phases. This is because it is not clearly seen, from the performance results of AISD, the contribution that the AIS makes.

Typically, the schedule length of a task graph generated by a scheduling algorithm is used as the main performance measure of the algorithm. The performance metric used for the comparison is the *normalized schedule length* (NSL). The normalized schedule length is defined to be schedule length obtained by a particular algorithm over schedule length obtained by the HEFT algorithm.

As implied in Section 4.1 the performance of AISD tends to be influenced by its input parameters, i.e., *minG, maxG, c, sr and λ*. The actual values of these parameters used for our experiments are: (1) minG = 3, (2) maxG = 10, (3) sr = 30%, (4) c = 10 and (5) λ = 2. It should be noted that they are chosen with the time complexity in mind.

5.1. Experiment Configuration

The parameters used in the experiments are summarized in Table 1. The total number of experiments conducted with various both randomly generated and real-world application task graphs on the five different numbers of processors is 84,000. More specifically, the random task graph set consists of 210 base task graphs generated with combinations of 10 graph sizes, 7 CCRs and 3 processor heterogeneity settings. For each combination 20 task graphs are randomly generated retaining the base one's characteristics. These 4,200 graphs are then experimented on with 5 different numbers of processors. Furthermore, each of the three applications is experimented on with the same number of task graphs (i.e., 21,000); hence the figure 84,000.

Table 1. Experimental parameters

Parameter	Value
The number of tasks	U(10, 600)
CCR	{0.1, 0.2, 0.5, 1, 2, 5, 10}
The number of processors	{2, 4, 8, 16, 32}
Processor heterogeneity	{100, 200, random}

The computation and communication costs of the tasks in each task graph were randomly selected from a uniform distribution with the mean equal to the chosen average computation and communication costs. The processor heterogeneity value of 100 is defined to be the percentage of the speed difference between the fastest processor and the slowest processor in a given system. For the well-known application task graphs, the matrix sizes and the number of input points are varied, so that the number of tasks can range from about 10 to 600.

Although the experiments are carried out with seven different CCRs as stated in Table 1 only experimental results obtained with three significant CCRs of 0.1, 1 and 5 are presented. This is due to the fact that these results are good enough to represent the performance of the three heuristics (AISD, HEFT and DBUS) for three fundamental task graph types (computation-intensive, moderate, communiation-intensive). What is more, the rest of the test results obtained from the task graphs with CCRs of 0.2, 0.5, 2 and 5 tend to be similar to those obtained from the task graphs with close CCRs. For instance, the test result acquired from the task graphs with CCR 5 does not show significant difference from the test result acquired from the task graphs with CCR 10.

5.2. Experimental Results

It is clearly shown in Figures 6 and 7 that the AISD algorithm delivers quite competitive schedule lengths irrespective of different application and system characteristics, e.g., graph sizes and the number of processors. The schedule lengths obtained from communication intensive task graphs indicate that the AISD algorithm is better suited for task graphs consisting of fine-grain tasks with large communication costs. This is also true for the DBUS algorithm. However, its performance drops noticeably with computation intensive task graphs. Moreover, when the number of processors in a given system is small even DBUS poorly performs with communication intensive task graphs. More precisely, the larger the ratio between the number of tasks and the number of processors, the poorer DBUS performs. This is due to the fact that a task with a single child may have to be assigned to more than one processor if the child task has been duplicated multiple times in order to cover the multiple child tasks of its own. This takes place recursively leading to increasing the schedule length significantly. The AISD algorithm, however, overcomes this drastic degradation by performing initial scheduling and task duplication separately. In other words, task duplication only takes place after all tasks in a task graph are scheduled without duplication.

Two major sources of the performance gain of AISD are its clonal selection and task duplication schemes as shown in Figures 6 and 7. The former refines and leads to decent quality schedules, and the latter makes efforts towards reducing the communication overhead.

The average schedule length of AISD computed based on communication intensive task graphs in the randomly generated task graph set, shown in Figure 6, is 18% on average and 32% at best, smaller than that of HEFT. The results on task graphs of the well-known applications in Figure 7 reconfirm this superior performance of AISD.

Although DBUS tends to deliver significantly smaller schedule lengths than those of HEFT for communication intensive task graphs its applicability is limited to those systems consisting of a large number of processors.

6. Conclusion

In this paper, we have presented a novel scheduling algorithm, called the AISD algorithm, for heterogeneous

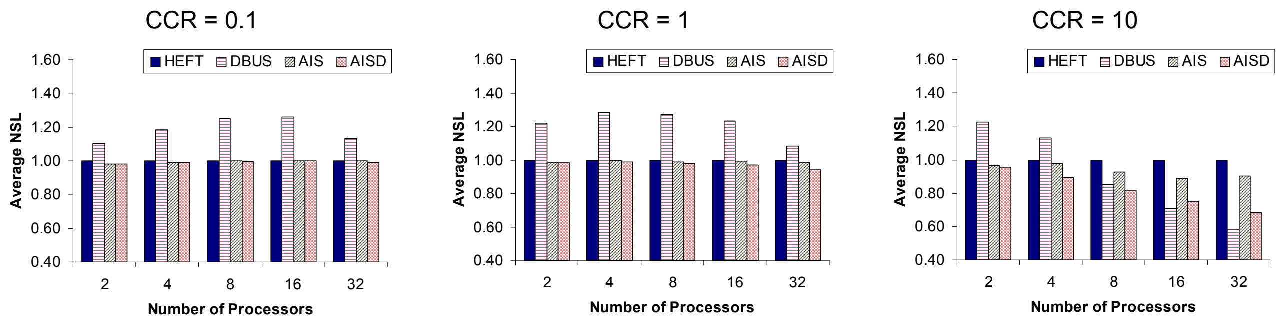
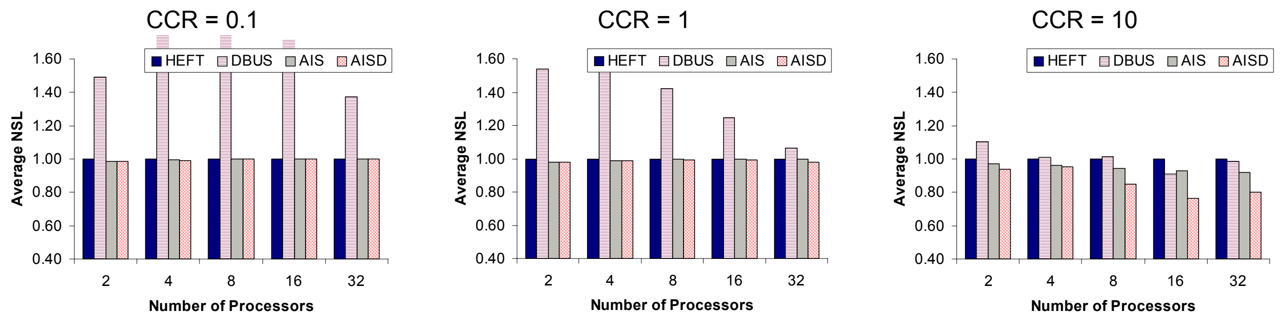
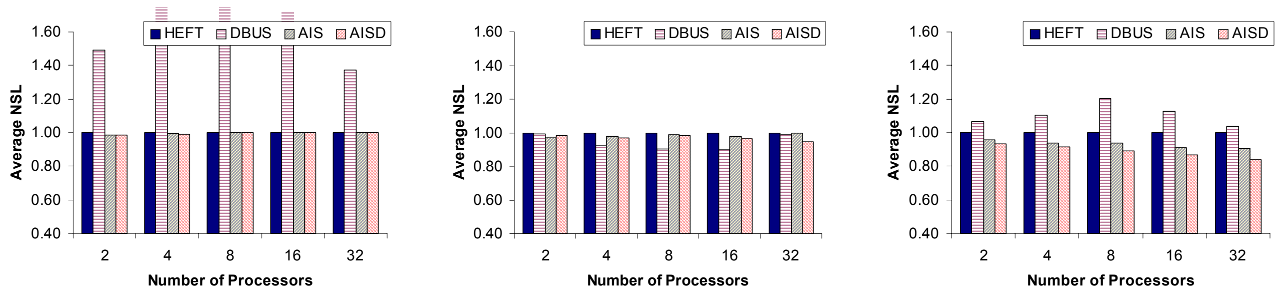


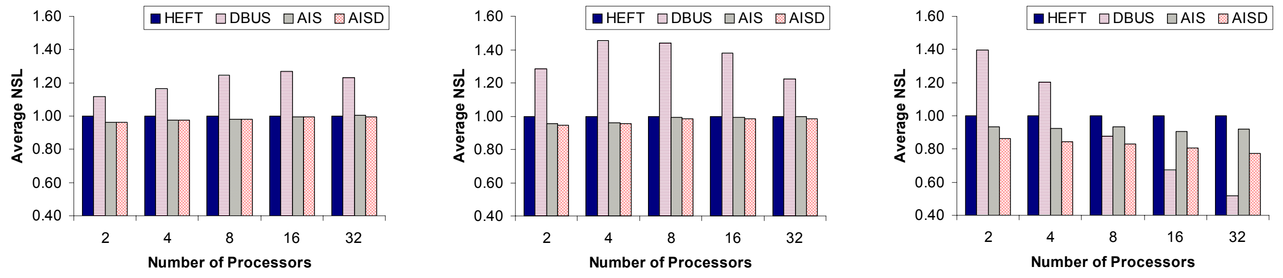
Figure 6. Average NSL of random DAGs



(a)



(b)



(c)

Figure 7. Average NSL for DAGs of (a) Laplace, (b) LU and (c) FFT

computing systems. AISD incorporates the clonal selection in the vertebrate immune system and task duplication, as its core components, into its scheduling. It is proved that these two irrefutably contribute to the superior performance of the proposed algorithm. Based on the performance results obtained from extensive experiments conducted with a comprehensive set of both randomly generated and well-known application task graphs and various system configurations, AISD consistently outperformed the two existing algorithms by a noticeable margin, especially when scheduling communication intensive task graphs.

References

[1] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., pages 238–239, 1979.

[2] L.J. Gonzales and J. Cannady. A self-adaptive negative selection approach for anomaly detection. In *Proceedings of Congress on Evolutionary Computation*, volume 2, pages 1561–1568, Jun. 2004.

[3] J. Kim and P.J. Bentley. Towards an artificial immune system for network intrusion detection: an investigation of clonal selection with a negative selection operator. In *Proceedings of Congress on Evolutionary Computation*, volume 2, pages 1244–1252, May 2001.

[4] T. Stibor, J. Timmis and C. Eckert. On the appropriateness of negative selection defined over Hamming shape-space as a network intrusion detection system. In *Proceedings of Congress on Evolutionary Computation*, volume 2, pages 995–1002, Sep. 2005.

[5] A. Swiecicka, F. Seredynski, and A.Y. Zomaya. Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. *IEEE Transactions on Parallel and Distributed Systems*, 17(3):253–262, Mar. 2006.

[6] A. M. Costa, P. A. Vargas, F. J. Von Zuben and P. M. Franca. Makespan minimization on parallel processors: an immune-based approach. In *Proceedings of Congress on Evolutionary Computation*, volume 1, pages 920 – 925, May 2002.

[7] Z. X. Ong, J. C. Tay and C. K. Kwoh. Applying the Clonal Selection Principle to Find Flexible Jop-Shop Schedules. In *Proceedings of International Conference on Artificial Immune Systems (ICARIS)*, pages 442–455, Aug. 2005.

[8] O. Engin and A. Doyen. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20(6):1083–1095, Aug. 2004.

[9] F. M. Burnet. The Clonal Selection Theory of Acquired Immunity. Cambridge University Press, 1959.

[10] L. N. de Castro and J. Timmis. Artificial Immune Systems: A New Computational Intelligence Approach, 1st ed., Springer-Verlag, London, page 58, 2002.

[11] S. Garrett. How Do We Evaluate Artificial Immune Systems?. *Evolutionary Computation*, 13(2): 145–177, Jun. 2005.

[12] M.-Y. Wu and D.D. Gajski. Hypertool: A Programming Aid for Message-Passing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, Jul. 1990.

[13] R.E. Lord, J.S. Kowalik, and S.P. Kumar. Solving Linear Algebraic Equations on an MIMD Computer. *Journal of the ACM*, 30(1):103–117, Jan. 1983.

[14] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to Algorithms. MIT Press, 1990.

[15] H. Topcuoglu, S. Hariri and M. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, Mar. 2002.

[16] D. Bozdag, U. Catalyurek and F. Ozguner. A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2005.