A Hybrid Heuristic-Genetic Algorithm for Task Scheduling in Heterogeneous Multi-core System

Chuan Wang, Jianhua Gu, Yunlan Wang, and Tianhai Zhao

School of Computer, NPU HPC Center, Xi'an, China
wangnwpu@163.com,
{gujh,wangyl,zhaoth}@nwpu.edu.cn

Abstract. Task scheduling on heterogeneous multi-core systems is NP-complete problem. This paper proposes a novel hybrid static scheduling algorithm named Hybrid Successor Concerned Heuristic-Genetic Scheduling (HSCGS) algorithm. The algorithm is a combination of heuristic and genetic scheduling algorithm. In the first phase we propose a heuristic algorithm named Successor Concerned List Heuristic Scheduling (SCLS) to generate a high quality scheduling result. SCLS algorithm takes the impact of current task's scheduling to its successor into account. The second phase implements an Improved Genetic Algorithm (IGA) for scheduling, to optimize the scheduling results of SCLS iteratively. The comparison experiments are based on both random generated applications and some real world applications. The performance of HSCGS is compared with some famous task scheduling algorithms, such as HEFT and DLS. The results show that HSCGS is the best of them, and the advantages go up with the increase of the heterogeneous factor of inter-core link bandwidth.

Keywords: Heterogeneous multi-core, Task scheduling, Heuristic algorithm, Genetic algorithm, Hybrid Scheduing, Directed acyclic graph.

1 Introduction

By integrating a number of cores with different capability together, heterogeneous multi-core architecture provides better performance compared with homogeneous system [1]. In order to make good use of heterogeneous multi-core system, tasks of parallel applications should be scheduled appropriately. The quality of task schedule has great impact on application execution efficiency. Traditional task scheduling algorithms are mostly designed for homogeneous systems and are difficult to achieve desired results on heterogeneous systems. The development of heterogeneous multi-core system brings new challenges to task scheduling algorithm. Task scheduling problem on heterogeneous multi-core systems is NP-complete [2].

Task scheduling algorithms are generally divided into static scheduling and dynamic scheduling [3]. In static scheduling algorithm, all information needed for scheduling must be known in advance, including the execution time of each task, the characteristics of the cores, and the amount of data passed between tasks and so on.

Static scheduling make decisions in compile time which does not increase the running cost of the application. Dynamic scheduling algorithms gather information needed for scheduling and make scheduling decisions at run time which can adapt to different circumstances.

Because of its key importance, task scheduling algorithm has been studied extensively, and a lot of static scheduling algorithms [2] have been proposed. These algorithms can be divided into three categories: heuristic algorithm, guided random search algorithm, hybrid of the previous two kinds of algorithms. List-based heuristic algorithm is a widely used heuristic algorithm, which can generate high quality task schedule in a short time, but it will not perform well in all cases. Genetic algorithm (GA) is the most widely used guided random search techniques. It aims to obtain near-optimal task schedules by imitating the process of biological evolution. Genetic Algorithm can usually get good scheduling results after sufficient number of generations, but its time complexity is much higher than heuristic algorithm [2]. Hybrid scheduling algorithms which combine heuristic algorithm and GA properly can overcome their shortcomings. Heuristic algorithm can offer high quality initial population for GA thereby decreasing the number of generations needed for getting good scheduling results.

In this paper, we propose a new hybrid static task scheduling algorithm, named Hybrid Successor Concerned Heuristic-Genetic Scheduling (HSCGS) algorithm. It is a two-phase algorithm for heterogeneous multi-core system. In the first phase, we propose a new list-based heuristic algorithm, named Successor Concerned List Scheduling (SCLS), to generate a high quality task schedule. Then, we propose an Improved Genetic Algorithm (IGA) in the second phase to optimize the task schedule generated by SCLS.

This paper is organized as follows: Section 2 describes task scheduling problem for heterogeneous system and related terminology; Section 3 introduces some task scheduling algorithms for heterogeneous systems; Section 4 gives a detailed description of the the HSCGS algorithm we proposed; Section 5 presents comparison experiments of HSCGS with H2GS, HEFT, CPOP and DLS, based on both random generated applications and some real world applications. In section 6, we present the conclusion of the whole paper and the plan of our future work.

2 Problem Description

Task scheduling on heterogeneous multi-core systems aims to minimize the schedule length(which is also called *makespan*), by assigning tasks to appropriate cores and arranging the execution order of tasks on each core without violating the precedence constraint between tasks. In general, static scheduling algorithms are based on the following assumptions:

1) The entire system is dedicated;
2) All cores in the computing system is fully-connected;
3) There is no link contention between cores;
4) Each task can run on any core in the system.

A parallel application is represented by a Directed Acyclic Graph (DAG), G = (V, E), where V represents the set of *v* tasks with, E is the set of *e* edges between tasks. V_i is the amount of data that task n_i need to process. Each $edge(i,j) \in E$ represents a precedence constraint between task n_i and task n_j, such that the execution of n_j cannot start before n_i finishes its execution. Data is a $v * v$ matrix, $data_{i,j}$ is the amount of data transported from the task n_i to task n_j. A task with no parent is called an entry task, and a task with no children is called an exit task.

Q represents the set of *q* cores of the heterogeneous system, Q_i represents the capability of core p_i. W is a $v * q$ matrix, $w_{i,j}$ indicates the execution time of task n_i on core p_j.

$$w_{i,j} = V_i / Q_j \tag{1}$$

The average execution time of task n_i is $\overline{w_i}$ as defined in (2).

$$\overline{w_i} = \sum_{j=1}^{q} w_{i,j}/q \tag{2}$$

The bandwidth of links between cores is represented by a $q * q$ matrix B. Communication initialization time is represented by array L with size *q*. The communication cost between task n_i and n_j is $c_{i,j}$ as defined in (3) (n_i is allocated on core p_m, n_j is allocated on core p_n).

$$c_{i,j} = L_m + \frac{data_{i,j}}{B_{m,n}} \tag{3}$$

$\overline{c_{i,j}}$ represents the average time to transfer data from task n_i to n_j.

$$\overline{c_{i,j}} = \overline{L} + \frac{data_{i,j}}{\overline{B}} \tag{4}$$

$\overline{L}$ represents the average communication initialization time of all the cores, $\overline{B}$ represents the average bandwidth of all the links.

Then we define two important attributes for heuristic algorithm, $EST(n_i, p_j)$ (earliest start time of task n_i on core p_j) and $EFT(n_i, p_j)$ (earliest finish time of task n_i on core p_j).

$$EST(n_{entry}, p_j) = 0 \tag{5}$$

$$EST(n_i, p_j) = \max\{ avail[j], \max_{n_m \in pred(n_i)}\{AFT(n_m) + c_{m,j}\} \} \tag{6}$$

$$EFT(n_i, p_j) = w_{i,j} + EST(n_i, p_j) \tag{7}$$

n_{entry} is the entry task of the application, $avail[j]$ is the first available time at which core p_j is ready for task execution, $pred(n_i)$ is the set of parents of task n_i. When a task n_m is assigned to a core, its start time and finish time is certain, and represented by $AST(n_m)$ and $AFT(n_m)$ [2].

After all tasks are scheduled, the finish time of exit task $AFT(n_{exit})$ is the finish time of the total application, named makespan. If there are more than one exit tasks, the makespan is the maximum.

$$makespan = \max\{AFT(n_{exit})\} \tag{8}$$

The primary goal of task scheduling is minimizing the makespan of the application.

3 Related Work

The quality of task schedule has great impact on the performance of applications running on the heterogeneous multi-core system. Some scheduling algorithms that support heterogeneous multi-core systems have been proposed, such as Heterogeneous Earliest Finish Time (HEFT) [2], Critical Path on a Processor (CPOP) [2], Longest Dynamic Critical Path (LDCP) Algorithm [4], Dynamic Level Scheduling (DLS) [8] and Hybrid Heuristic–Genetic Scheduling (H2GS) [3].

In this section, we describe 3 static scheduling algorithms (HEFT, DLS and H2GS) for heterogeneous systems in detail.

3.1 Heterogeneous Earliest Finish Time Algorithm

The HEFT algorithm has two major phases: a task prioritizing phase for computing the priorities of all tasks and a core selection phase for scheduling the task with the highest priority to the "best" core, which minimizes the task's finish time [2].

The priority of task n_i is equal to its upward rank ($rank_u$) value as defined in (9).

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)}\{\overline{c_{i,j}} + rank_u(n_j)\} \tag{9}$$

Upward rank is the length of the critical path from current task to the exit task, including the tasks' average running cost and average communication cost. $succ(n_i)$ is the set of immediate successors of task n_i. After the computation of each task's priority, all the tasks are pushed into a task queue in decreasing order of priorities. Tie-breaking is done randomly.

At each step of the core selection phase, the task with the highest priority is selected and assigned to the core that minimizes the EFT value using the insertion-based policy.

The time complexity of HEFT algorithm is $O(q * e)$ when there are q cores in the heterogeneous system and e edges in the application DAG. The experiment results in paper [2] indicate that HEFT is better than CPOP, DLS, MH [12] and LMT [13] with different DAG size and CCR value. HEFT only concerns the EFT of the header task in the priority queue in core selection phase, and this may not perform well in some cases (Fig. 2).

3.2 Dynamic Level Scheduling Algorithm

DLS algorithm uses dynamically-changing priorities to match tasks with processors. Scheduling decision is made based on Dynamic Level (DL) of each [task, core] pair. For task n_i, core p_j, the DL value is defined as (10) and (11).

$$DL(n_i, p_j) = rank_{ur}(n_i) - EST(n_i, p_j) \tag{10}$$

$$rank_{ur}(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)}\{rank_{ur}(n_j)\} \tag{11}$$

At each step of the algorithm, DLS computes the DL value of every [task, core] pair if the task was not scheduled. The pair $[n_i, p_j]$ that has the largest DL value is selected for scheduling.

The time complexity of DL algorithm is $O(q * v^3)$, when there are q cores and v tasks. The experiment results in paper [8] indicate that DL algorithm is better than MH and LMT.

3.3 Hybrid Heuristic–Genetic Scheduling Algorithm

H2GS algorithm is a hybrid scheduling algorithm. Firstly, H2GS use a heuristic algorithm named LDCP [4] to generate a near-optimal scheduling result. Then, H2GS optimize the scheduling result generated by LDCP iteratively using Genetic Algorithm for Scheduling (GAS) [3].

GAS is a genetic algorithm for task scheduling. In GAS, a two-dimensional string chromosome is used to encode the task schedule. Customized genetic operators are developed specifically for the task scheduling problem to maintain population diversity and simultaneously enable an efficient stochastic search process [3]. GAS uses 1/makespan as the fitness value for selection mechanism. The fittest 10% of the chromosomes in the population are copied without change to the next generation. GAS algorithm runs for a predetermined number of generations which is set according to the characteristics and real-world context of the parallel application [3].When the GAS algorithm finished, H2GS select the scheduling result which has the largest fitness value as the final task schedule.

The experiment results in paper [3] shows that H2GS gets a better task scheduling result than HEFT and DLS, but it takes longer time to generate the result. GAS does not take care of the the regeneration phenomenon (generating similar chromosomes) which leads to premature convergence [15].

4 The HSCGS Algorithm

In this section, we describe the HSCGS algorithm in detail. HSCGS is a hybrid static task scheduling algorithm for heterogeneous multi-core system. In the first phase we propose a list heuristic algorithm, named Successor Concerned List Scheduling

(SCLS), to generate a near-optimal scheduling result in a short time. Then, we propose an Improved Genetic Algorithm (IGA) to optimize the scheduling result generated by SCLS algorithm.

4.1 Successor Concerned List Scheduling Algorithm

SCLS (Fig. 1) is a list heuristic scheduling algorithm that has two phases. Firstly, SCLS computes the priority of each task and pushes them in a task queue by decreasing order of their priorities. Then, pop the task with the highest priority and schedule it on the most appropriate core. SCLS cares about not only current task's upward rank but also the impact on its successors. SCLS schedules tasks in a more global perspective in order to minimize the makespan of the application.

```
Compute priorities for all task as equation (12);
Push all tasks into a task queue by decreasing order of priorities;
While there are unscheduled tasks in the task queue do
    Select the task $n_i$ which has the highest priority;
    If $n_i$ has unscheduled parents
        Select the parent $n_j$ which has the highest priority as the selected task;
    Else
        Set $n_i$ as the selected task;
    For each core in the system do
        Compute appropriate value as equation (13);
    Assign the selected task to the core with the minimum appropriate value;
End While
```

Fig. 1. The SCLS algorithm

Task Prioritizing Phase: In this phase, the priority of each task is set with a combination of normalized upward rank and numbers of successors. SCLS concerns not only upward rank of current task but also how many successors it has. For parallel applications, scheduling tasks with more successors early can improve efficiency apparently. SCLS computes the priority of task n_i as (12).

$$\mathrm{Priority}(n_i) = \frac{\mathrm{rank}_u(n_i)}{\mathrm{rank}_u(n_{entry})} + \frac{\mathrm{succ}_i}{v} \tag{12}$$

n_{entry} is the entry task of the application, $succ_i$ is the number of n_i's successors, *v* is the total number of tasks.

When all tasks' priorities are computed, push them into a task queue in decreasing order of their priorities.

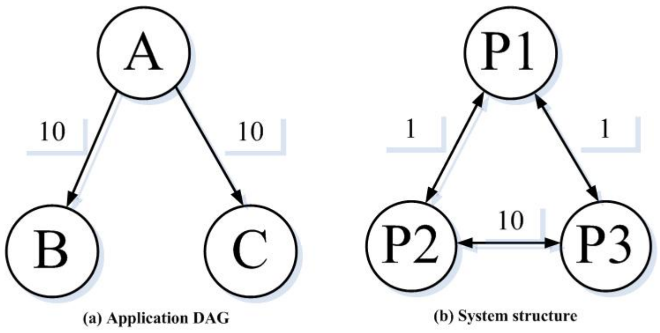
Core Selection Phase: At each step of this phase, SCLS selects the task with the highest priority if it has no unscheduled parent, and assigns it to the core which minimizes the appropriate value. If the task has one or more unscheduled parents, SCLS selects the one with the highest priority and assign it to the core which minimizes the appropriate value. The appropriate value of a core p_m is computed as (13).

$$\text{Appropriate}(n_i, p_m) = \text{Min}(\,\text{EFT}(n_i, p_m) + \text{Max}_{n_j \in succ(n_i)}(\,\overline{c}_{m\,(i,j)}\,)\,) \quad (13)$$

$$\overline{c}_{m\,(i,j)} = L_m + \frac{data_{i,j}}{\overline{B}_m} \quad (14)$$

$$\overline{B}_m = \sum_{i=1\ldots q} B_{m,i} \,/\, q \quad (15)$$

n_i is the selected task that to be scheduled, $succ(n_i)$ is the set of successors of task n_i. $\overline{B}_m$ is the average bandwidth of links between core p_m and other cores. L_m is the communication initialization time on core p_m.



Computation Cost	P1	P2	P3
A	2	3	4
B	5	2	2
C	6	2	2

(c) Computation cost matrix

Fig. 2. An example of Core Selection

Fig. 2 shows a circumstance that the core selection strategy of SCLS is better than algorithms that only cares about the EFT value (such as HEFT). Fig.2 (a) is the application DAG, the value on the edge shows the size of data communications between tasks. Fig.2 (b) is the system structure and the value on the edge is the bandwidth of the link. Fig2(c) is the computation cost matrix. The HEFT scheduling result is {A—>P1, B—>p1, C—>P1}, and the makespan is 2+5+6=13. The SCLS scheduling result is {A—>P2, B—>p2, C—>P3}, and the makespan is max {3+2, 4+2} =6.

4.2 Improved Genetic Algorithm

This section proposes the Improved Genetic algorithm (IGA) and describes how to use it to optimize task scheduling result. IGA add a preprocessing process before the selection phase to avoid the regeneration phenomenon (generating similar chromosomes) which leads to premature convergence [15]. IGA uses a three-phase selection mechanism, in order to maintain the diversity of the population and optimize the population at the same time.

Schedule Encoding and Chromosome Decoding: The ways of coding and decoding have great influences on crossover and mutation operation. In order to decrease the complexity of IGA, we use a simple one-dimensional integer chromosome G to encode a task schedule. The length of G is *v*, which equals to the number of tasks of the application. The index of the array represents the task ID, and the value at each index represents the core ID that the task assigned to. For example, $\mathrm{G[i]} = \mathrm{j}$ means assigning task $\mathrm{n_i}$ to core $\mathrm{p_j}$. There is no priority information in the chromosome. The priorities of tasks are the same as those of SCLS algorithm.

At each step of the decoding phase of chromosome G, IGA selects the head task $\mathrm{n_i}$ from the task queue if $\mathrm{n_i}$ has no unscheduled parent, and assigns $\mathrm{n_i}$ to core $\mathrm{P_{G[i]}}$. If $\mathrm{n_i}$ has unscheduled parents, IGA selects the task $\mathrm{n_m}$ which has the highest priority and assigns it to core $\mathrm{P_{G[m]}}$. The decoding phase ends until the queue is empty. The relationship between task schedule and chromosome is one-on-one.

Initialization: In initialization phase, IGA constructs the initial population. The size of the population is N. The scheduling result of SCLS is encoded and added into the initial population. The rest of the initial population is generated randomly (set $\mathrm{G[i]}$ with a random integer between 1 and *q*, *q* is the number of cores).The minimum size of population is *q*.

Fitness Evaluation: The fitness value represents the quality of the chromosome. IGA evaluates fitness value as (16).

$$\text{fitness} = 1/\text{makespan} \tag{16}$$

Crossover and Mutation

1) Crossover: IGA uses a simple two-point swap crossover operator. Firstly, IGA selects two chromosomes from the chromosome pool randomly. Then randomly select two points and exchange the subsequences of each selected chromosome between the two points. Finally, add the new generated chromosomes to the chromosome pool. This process is performed with probability P_c.

2) Mutation: The mutation works on a single chromosome in the chromosome pool. The mutation operation selects a chromosome and a mutation point randomly, and then changes the value to a random integer from 1 to q. Then add the new generated chromosome to the chromosome pool. This process is performed with probability P_m.

Preprocessing: Some new chromosomes which are similar with existing ones will be generated in the process of IGA (called regeneration phenomenon). This may cause the search process traps in a local optimum [15]. IGA adds the preprocessing phase to avoid this. Fig. 3 shows the preprocessing algorithm. This process is performed with probability P_w, $P_w < (P_c + P_m)$.

```
Select two chromosomes G1 and G2 from the chromosome pool randomly;
Counter = 0;
v is the length of the chromosomes;
T is the threshold defined in advance;
For i = 0 to v-1 do
    If G1[i] = G2[i] do
            Counter = Counter+1;
    End If
End For
S = Counter / v ;
If S > T do
    Calculate the fitness of G1 and G2;
    Remove the chromosome which has the smaller fitness value;
End If
```

Fig. 3. The preprocessing algorithm

Selection: In selection phase, IGA selects N chromosomes from the chromosome pool as the population of next iteration using a three-phase selection mechanism.

In the first stage, IGA compares the new chromosomes generated by crossover and mutation with their parents. If a new chromosome's fitness value is bigger than its parents', IGA moves it to the new population. Assuming there are N_1 chromosomes in the new population after this stage. In the second stage, IGA sets N_2 as $N - N_1$, and moves the fittest $N_2 * 20\%$ chromosomes to the new population. In the third stage, move $N_2 * 80\%$ chromosomes selected randomly to the new population.

After the selection phase, there are N chromosomes in the new population. Then the next iteration begins.

Termination: There are several termination strategies to stop the iteration. IGA runs for a predetermined number of generations which is set according to the characteristics and real-world context of the parallel application [15].

5 Experiment Results and Discussion

In this section, the performance of HSCGS is presented in comparison with H2GS, HEFT, CPOP, and DLS on our simulation framework. We consider two sets of benchmark application graphs as the workload for testing the algorithms: randomly generated DAGs and DAGs that represent some of the real world problems.

First, we present the performance metrics used for evaluating the quality of these scheduling algorithms. Then, we describe the mechanism for generating random application DAGs and the experimental parameters. At last, we present the experimental results.

5.1 Performance Metrics

The metrics we used includes Normalized Schedule Length (NSL) [16], Speedup [2] and Efficiency of the algorithms [3].

Normalized Schedule Length (NSL): The NSL of a task schedule is defined as the makespan to the lower bound of the schedule length. It is calculated as (17).

$$\mathrm{NSL} = \mathrm{makespan}/\min_{j\in(1\ldots p)}\left(\sum_{n_i\in CP} w_{i,j}\right) \tag{17}$$

p is the number of cores. CP is the critical path of the application DAG. The lower bound of the schedule length is the total cost of tasks on the CP when they are assigned to the fastest core.

Speedup: The speedup of a task schedule is the ratio of the serial schedule length obtained by assigning all tasks to the fastest core, to the makespan of task schedule.

$$\mathrm{Speedup} = \min_{j\in(1\ldots p)}\left(\sum_{i=1}^{v} w_{i,j}\right)/\mathrm{makespan} \tag{18}$$

v is number of tasks in the DAG.

Efficiency: The efficiency of a task schedule is defined as the speedup to the number of cores.

$$\text{Efficiency} = \text{Speedup}/\ q \tag{19}$$

We use the average value of these metrics over a set of DAGs to compare the quality of the scheduling algorithms.

5.2 Generating Random DAGs

We use a synthetic DAG generator to create random application DAGs. The synthetic DAG generator uses a set of input parameters to determine some characteristics of the random DAGs.

1) N= {20, 40, 60, 80,100}, the total number of tasks in each DAG.
2) Mindata =2048, the minimum size of data processed by a task in Flop.
3) Maxdata=11268, the maximum size of data processed by a task in Flop.
4) Fat= {0.1, 0.2, 0.8}, width of the DAG, that is maximum number of tasks that can be executed concurrently. A small value will lead to a thin DAG with a low task parallelism, while a large value induces a fat DAG with a high degree of parallelism.
5) Density= {0.2, 0.8}, the numbers of dependencies between tasks of two consecutive DAG levels.
6) CCR= {0.2, 0.5, 1, 2, 5}, communication to computation cost ratio, the average communication cost divided by the average computation cost of the application DAG.

5.3 Experimental Parameters

The characteristics of the simulated heterogeneous computing system and parameters of the IGA algorithm we used in our experiment are as below.

1) Number of cores q={2, 4, 8, 16, 32}
2) Heterogeneity factor of cores $P_h = \{2, 4, 8, 16\}$
3) Heterogeneity factor of links' bandwidth between cores $B_h = \{1, 5,10,15\}$
4) $P_c = 0.7$
5) $P_m = 0.5$
6) $P_w = 0.8$
7) Population size of IGA $\text{PopSize} = 2 * q$
8) Number of generations of IGA $\text{NG} = \{5, 10, 15, 20\}$

5.4 Performance Results and Discussion

The results are presented over two kinds of application DAGs: randomly generated DAGs and DAGs that represent Fast Fourier Transformation and Gauss elimination algorithm.

Results of Randomly Generated DAGs: We create 300 random DAGs over each value of DAG size (so, there are 1500 DAGs in total), and calculate the average NSL and speedup of task schedules generated by HSCGS, H2GS, HEFT, CPOP and DSL.

The results are presented by Fig. 4 and Fig. 5. The NSL based performance ranking is {H2GS, HSCGS, HEFT, DLS, CPOP}, and the average NSL value of H2GS is nearly equal with that of HSCGS. The average NSL value of HSCGS is better than HEFT by 7%, DLS by 10% and CPOP by 12%. The average speedup value of HSCGS is better than HEFT by 4.7%, DLS by 6% and CPOP by 10%.

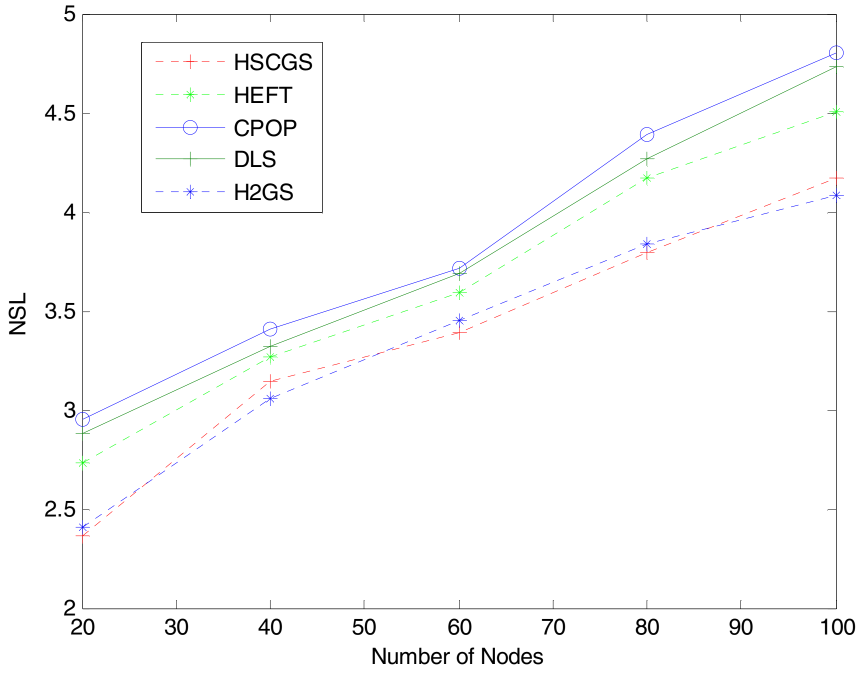


Fig. 4. Average NSL of random DAGs

Results of Fast Fourier Transformation: A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. The DAG of the FFT algorithm is characterized by the size of the input vector. If the vector size is M, the number of tasks is $(2 * M - 1)(M * \log_2 M)$. Fig. 6(b) shows the DAG of FFT with input vector size of 4.

In our experiment, the input vector size ranges from 2^1 to 2^5. Fig. 7 presents the average NSL of these four scheduling algorithms over different vector size. The NSL based performance ranking is {H2GS, HSCGS, HEFT, DLS, CPOP}. The average NSL of HSCGS is almost the same as that of H2GS. Fig. 8 presents the average Efficiency of these four scheduling algorithms over different number of cores (range from 2^1 to 2^5). The Efficiency based performance ranking is the same as that of NSL.

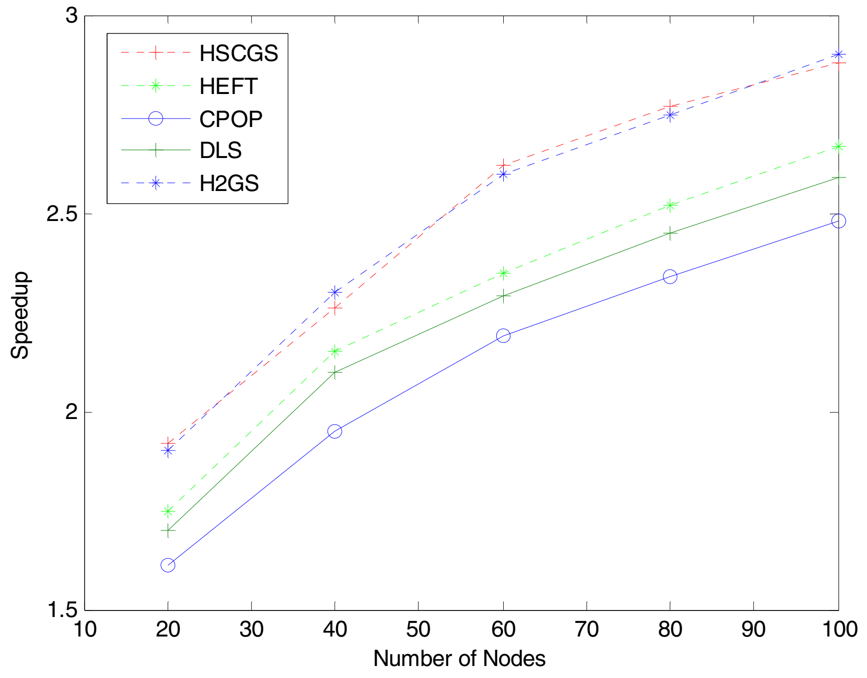


Fig. 5. Average Speedup of random DAGs

Results of Gauss Elimination Algorithm: In linear algebra, Gaussian elimination is an algorithm for solving systems of linear equations. It can also be used to find the rank of a matrix, to calculate the determinant of a matrix and the inverse of an invertible square matrix. The DAG of the Gauss elimination algorithm is characterized by the size of the input matrix. If the matrix size is $M * M$, the number of tasks is $(M^2 + M - 2)/2$. Figure 6(a) shows the DAG of Gauss elimination with input matrix size of 5.

The matrix size we used in the experiment ranges from 5 to 20, with an increment of 2 or 3. Fig. 9 presents the average NSL of these four scheduling algorithms over different matrix size. The NSL based performance ranking is {HSCGS, H2GS, HEFT, DLS, CPOP}. Fig. 10 presents the average Efficiency of these four scheduling algorithms over different number of cores (range from 2^1 to 2^5). The Efficiency based performance ranking is the same as that of NSL.

The extension of our experiment compares the average NSL of HSCGS, H2GS, HEFT, CPOP and DLS over different heterogeneity factor of links' bandwidth between cores($B_h = \{1, 5, 10, 15\}$). The result indicates that the superiority of HSCGS goes up, with the increase of B_h.

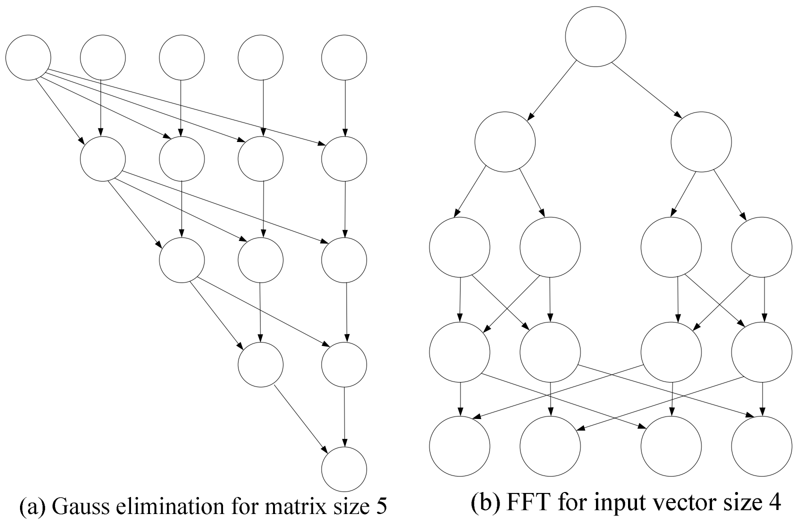

(a) Gauss elimination for matrix size 5
(b) FFT for input vector size 4

Fig. 6. DAG of Gauss elimination and FFT

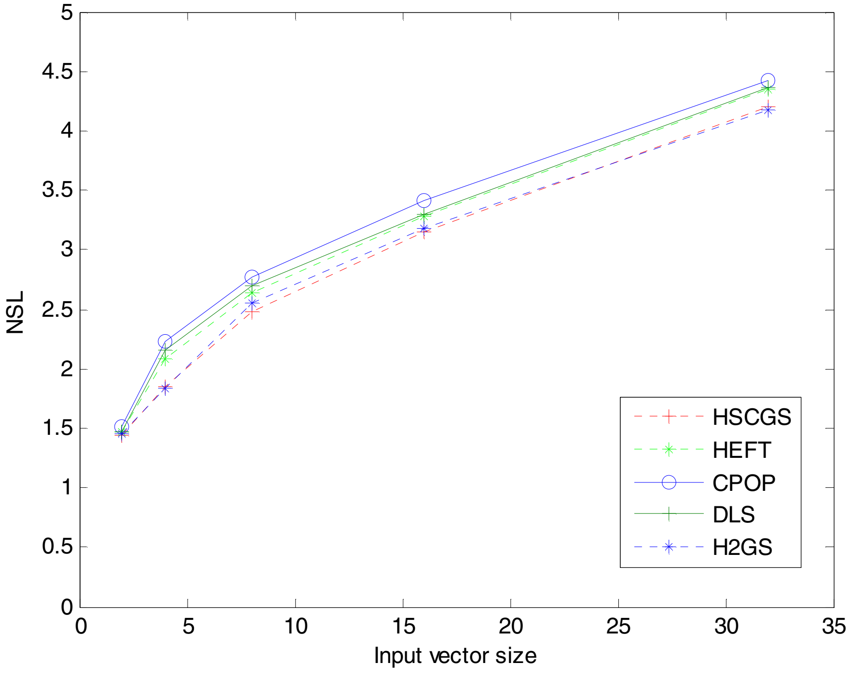


Fig. 7. Average NSL of FFT

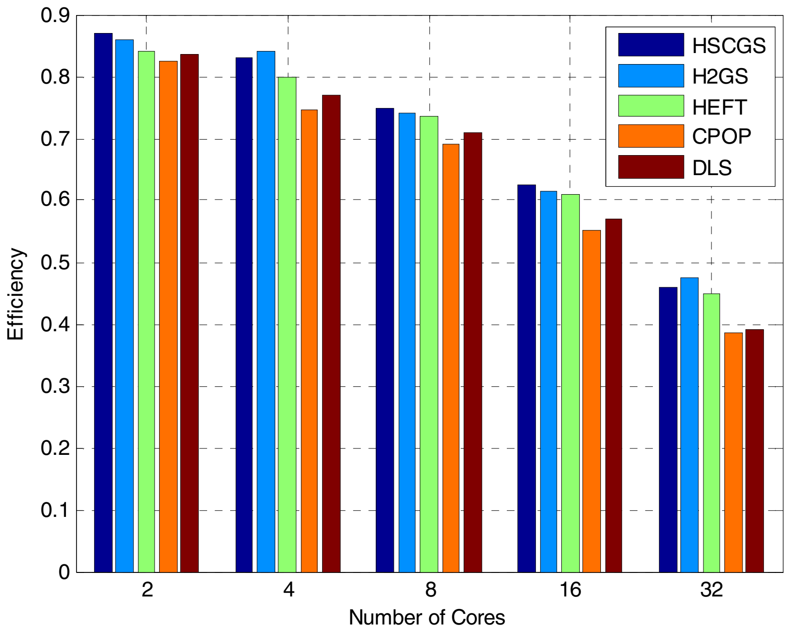


Fig. 8. Average Efficiency of FFT

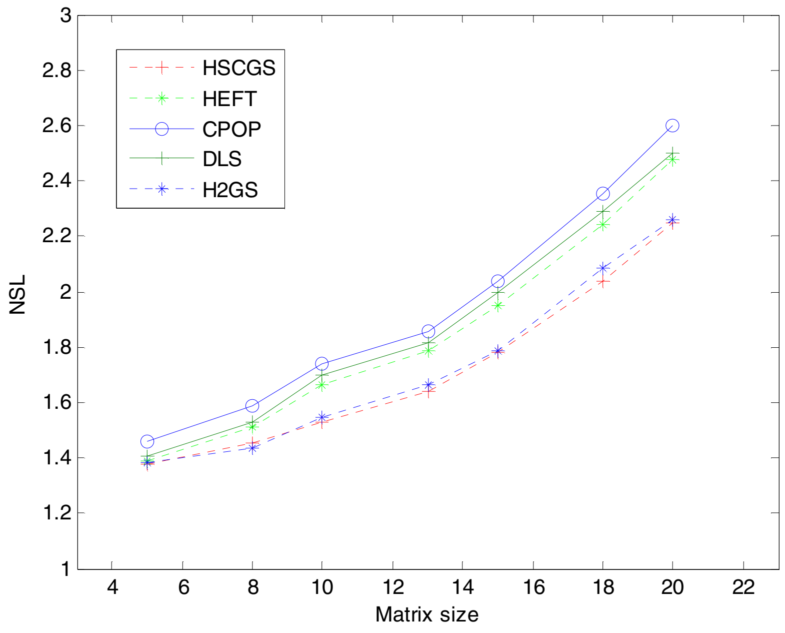


Fig. 9. Average NSL of Gauss elimination

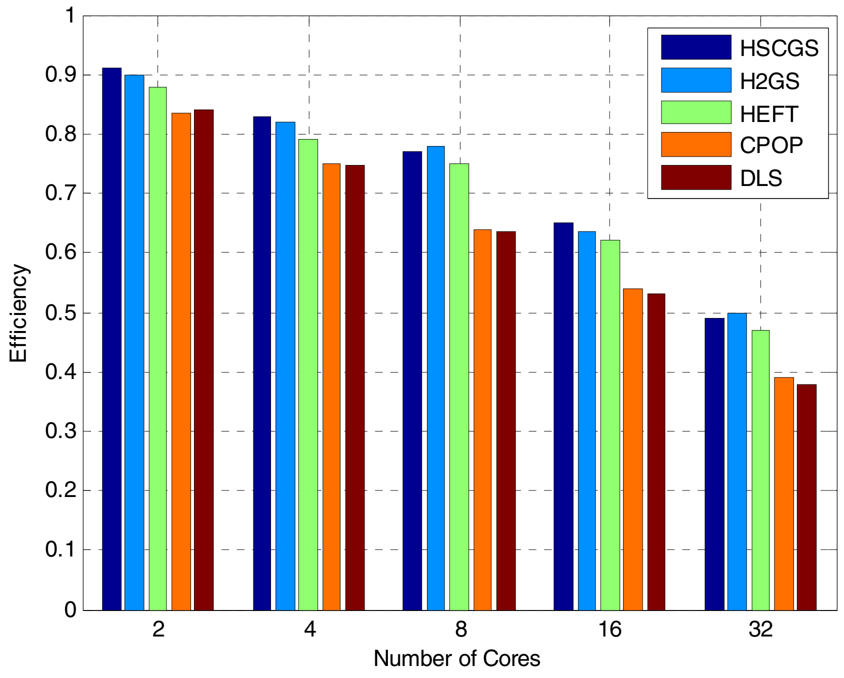


Fig. 10. Average Efficiency of Gauss elimination

6 Conclude

In this paper, we propose a novel hybrid static task scheduling algorithm named HSCGS for heterogeneous multi-core systems.

The algorithm is a combination of heuristic and genetic scheduling algorithm. In the first phase we propose a heuristic algorithm named Successor Concerned List Heuristic Scheduling (SCLS), to generate a near-optimal scheduling result. SCLS algorithm takes the impact of current task scheduling to its successors into account.

The second phase implements an Improved Genetic Algorithm (IGA) for scheduling, to optimize the scheduling result of SCLS iteratively. Before the selection phase of genetic algorithm, IGA adds a preprocessing phase to avoid the phenomenon of "regeneration" which leads to premature convergence by removing the chromosomes that almost the same as the existing ones. The selection phase of the IGA algorithm uses a three-phase selection mechanism, in order to maintain the diversity of the population and optimize the population at the same time.

The comparison experiments are based on both random generated applications and some real world applications (FFT and Gaussian elimination). The performance of HSCGS is compared with some famous task scheduling algorithms, H2GS, HEFT, CPOP and DLS. The results indicate that HSCGS is better than the other algorithms, and the advantages go up with the increase of the heterogeneous factor of inter-core links' bandwidth.

HSCGS is based on the assumption that all cores are fully-connected, but the heterogeneous distributed computing systems in the real world usually do not correspond with this assumption. So we plan to extend our algorithm on heterogeneous arbitrarily-connected systems and implement a task scheduling platform on top of StarPU [14].

Acknowledgements. This research is supported by the National High-tech R&D Program of China (863 Program) under Grand No.2009AA01Z142 and the Research Fund of Sugon.

References

1. Kumar, R., Tullsen, D., Jouppi, N., Ranganathan, P.: Heterogeneous Chip Multiprocessors. IEEE Computer, 32–38 (November 2005)
2. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Trans. Parallel and Distributed Systems 13(3), 260–274 (2002)
3. Daoud, M.I., Kharma, N.: A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. J. Parallel Distrib. Comput. 71, 1518–1531 (2011)
4. Daoud, M.I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput. 68, 399–409 (2008)
5. Wen, Y., Xu, H., Yang, J.: A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system. Information Sciences 181, 567–581 (2011)
6. Eswari, R., Nickolas, S.: Path-based Heuristic Task Scheduling Algorithm for Heterogeneous Distributed Computing Systems. In: 2010 International Conference on Advances in Recent Technologies in Communication and Computing (2010)
7. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surveys 31(4), 406–471 (1999)
8. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection constrained heterogeneous processor architectures. IEEE Trans. Parallel Distributed Systems 4(2), 175–187 (1993)
9. Kwok, Y.K., Ahmad, I.: Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Trans. Parallel Distributed Systems 7(5), 506–521 (1996)
10. El-Rewini, H., Lewis, T.G.: Scheduling parallel program tasks onto arbitrary target machines. J. Parallel Distributed Comput. 9(2), 138–153 (1990)
11. Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter control in evolutionary algorithms. Stud. Comput. Intell. 54, 19–46 (2007)
12. Ilavarasan, E., Thambidurai, P., Mahilmannan, R.: Performance effective task scheduling algorithm for heterogeneous computing system. In: Proc. 4th International Symposium on Parallel and Distributed Computing, France, pp. 28–38 (2005)
13. Iverson, M., Ozguner, F., Follen, G.: Parallelizing existing applications in a distributed heterogeneous environment. In: Proc. 4th Heterogeneous Computing Workshop, Santa Barbara, CA, pp. 93–100 (1995)

14. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A.: STARPU: a unified platform for task scheduling on heterogeneous multicore architectures. University of Bordeaux – LaBRI – INRIA Bordeaux Sud-Oues
15. Moghaddam, M.E., Bonyadi, M.R.: An Immune-based Genetic Algorithm with Reduced Search Space Coding for Multiprocessor Task Scheduling Problem. Int. J. Parallel Prog., doi:10.1007/s10766-011-0179-0
16. Bansal, S., Kumar, P., Singh, K.: An improved duplication strategy for scheduling precedence constrained graphs inmultiprocessor systems. IEEE Trans. Parallel Distrib. Syst. 14, 533–544 (2003)
17. Chung, Y.C., Ranka, S.: Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed-memory multiprocessors. In: Proc. Supercomputing 1992, Minneapolis, MN, pp. 512–521 (1992)
18. Wu, M., Dajski, D.: Hypertool: A programming aid for message passing systems. IEEE Trans. Parallel Distrib. Syst. 1, 330–343 (1990)