Xiaohong Kong,
Ruihua Li,
Yanqun Zhang

Henan Institute of Science & Technology, Xinxiang, China

SCHEDULING TASKS TO MULTI-PROCESSOR PLATFORM USING CONSTRAINT PROGRAMMING AND TABU SEARCH

Сяохун Кун,
Жуйхуа Лі,
Яньцюнь Чжан

Хенаньський інститут науки та технологій, Сіньсян, Китай

ПЛАНУВАННЯ ЗАДАЧ, ЩО СТОЯТЬ ПЕРЕД БАГАТОПРОЦЕСОРНОЮ ПЛАТФОРМОЮ, З ВИКОРИСТАННЯМ ПРОГРАМУВАННЯ В ОБМЕЖЕННЯХ І МЕТОДУ ПОШУКУ ІЗ ЗАБОРОНАМИ

Purpose. Multiple processors can be integrated together into a multi-core platform and deal with large-scale science problems because of its excess computation and extensive parallelization. This paper mainly aims at tasks scheduling problem on this platform with shared memory and communication channel.

Methodology. Generally, intensive-computation job is portioned into coarse-grain sub-tasks so that tasks are executed in parallel (simultaneously) to improve the computation performance. The proposed algorithm employs constraint programming to find a feasible solution and uses local search to impove the solution and speed up the process.

Findings. Sharing a variety of available resources, the multi-processors scheduling is a complex combinatory optimization problem and resource-constrainted problem. We investigate how to obtain a rational solution or sub-optimal solution in a short period. At the same time, tasks with different features are also investigated to find how the performances of applications are influenced in specific target platform.

Originality. When tasks are assigned to different processors, various resource constraints, including data storage, executing cost, tasks priority and communication cost among processors, must be considered. A hybrid algorithm based on Tabu search is studied to optimize the completion time of applicaitons satisfying these contraints.

Practical value. The algorithm is implemented in IBM ILOG CPLEX Optimization Studio environment and gives better results compared with other algorithms. The proposed algorithm is an effective strategy and the technique can improve search efficiency and solution performance for multiprocessor scheduling problem.

Keywords: *multi-processors, task scheduling, constraint programming, tabu search*

Introduction. Many applications require high-performance hardware, such as multimedia embedded systems, large-scale computing and medical imaging, etc. The speed of a single processor system cannot satisfy real-time requirements [1], so mixed hardware/software design and multi-processor on-chip (multi-core system) are a promising solution to this problem [2, 3]. In a multi-processor system, multiply processors, memory, input and output interfaces and communication bus are integrated together. Each processor has independent computing power and local storage for both instructions and data required by applications. Multi processors can run tasks simultaneously and achieve high throughput by parallelizing task execution and sharing resources. Generally, the IBM Cell Broadband Engine (Cell BE) is a typical example due to its super computation power and parallelization [3, 4].

In order to use the potential computation capacity, an application can be decomposed into different subtasks, and subtasks have precedence constraints and data dependence. Different subtasks can be executed in multi processors concurrently and the system completes jobs in less time. Due to communication delay between processors, double processors cannot halve the processing time. The performance relies on how correctly the application is allocated to processors and scheduled and the reasonable distribution of workload is important to speed up the processing. In determining when and how to distribute the workload and data, the allocating and scheduling algorithms are the key to be taken into account. The objective of the algorithms is to minimize the parallel completion time or schedule length by assigning the subtasks to the processors and scheduling them so as to satisfy the precedence constraints. However, the multi-processor platform is a heterogeneous architecture and it is still a challenge that an optimal solution can be obtained in polynomial time for allocating and scheduling.

Generally, the multiprocessor scheduling problem is NP-hard. A myriad of strategies have been investigated [3, 4] in Cell Broadband Engine. Considering resource limit, this problem is within typical resource-constrained optimization problems and constraint programming is an efficient strategy to solve such

problems [5, 6]. Meanwhile, a lot of optimization software is developed to solve these tough problems. ILOG optimization components provide modelling framework and solving strategy for the optimization problem. Its application kit can be used for all common computer platforms and optimization algorithms can be called at a variety of programming environment through the API interface [7]. A hybrid algorithm for multiprocessor scheduling is developed based on local search and the results demonstrate efficiency to scheduling.

Problem model statements. When applications run on the multi-core platform, the workload and data are partitioned among the available processor elements and memory.

The platform model. The target system is commonly assumed to consist of processors connected by an interconnection bus in which a message is transmitted through high bandwidth links. Here the platform is a heterogeneous system with dissimilar processors. When the task is executed the executing time is not equal in different processors. Every processor can run itself individual programs and has local memory for data. Additionally, there is a larger on-chip memory all the processors share and the memory is huge enough for all the data. The platform architecture enables fundamental improvement in processing performance and much scalable computation power.

The task graph. An application is typically modelled by a directed acyclic task graph (DAG) [8]. The task graph represents the decomposition of the application into different subtasks which are linked by communicated data. In a task graph, there are two kinds of symbol: nodes and arcs. The nodes $n_i \in N$ represent subtasks and the arc $a_{i,j} = (n_i, n_j) \in A$ defines the communication between the connected tasks. Each task has two related parameters, the computation data $comp(n_i)$ and the execution time $execu(n_i)$. The computation data takes a certain amount of storage space. If two nodes are connected by an arc, they have a communication from the parent task which produces the communication to the child task which consumes it. The sets of immediate parents and children of n_i are described by $parent(n_i)$ and $child(n_i)$ respectively. Each arc is associated with three relevant parameters, the communication data $comm(n_i, n_j)$ transferred from n_i to n_j, the reading time $rd(a_{ij})$ from parent nodes and the writing time $wr(a_{ij})$ to child nodes, and the communication data also requires a certain memory space. The consumers of the communication obtain data from the memory during the reading time and the producer of the communication transfer the data during the writing time. For each node, the number of the parent nodes is called its in-degree, and the number of child nodes its out-degree. Each communication is directed and the direction imposes precedence constraints between the tasks n_i and n_j, and dictates that task n_j cannot be started until n_i has been completed. Subsequently, the terms "node" and "task" are alternative and the same for "time" and "cost".

Every task starts at reading communicating data and must be executed consecutively. When computation data and communication data are assigned to different memory space, it affects the task execution time and communication time. If the computation data is assigned to remote memory or not the same processor to perform the task, it delays the execution time by latency multiplier. The situation is the same to communication data.

The application model. When tasks are to be scheduled on multiprocessors platform, there are some conventions. A processor can deal with one task and the task can be executed by one processor at a time. Each task occupies the processor and requires communication channel and storage space at the same time. All of the computation data and the communication data can be assigned to local storage of a processor, or the shared memory. Considering the multiprocessor architecture, each processor on the platform can access shared memory and local memory. When the data is read from or written to the shared memory, the cost is greater than the time accessing local memory. This difference is reflected in the task graph by setting the correct values for the communication overhead on the relevant arcs. In order to measure the delay due to different storage memory, the delay factor l_{comp}, l_{comm} is defined so that the execution time of task n_i on p_j ($p_j \in P$)(not the processor computation data is assigned) is given by $execu(n_i) * l_{comp}$. When two adjacent tasks n_i and n_j are allocated to distinct processors which communication a_{ij} data is assigned to, the cost associated with the communication a_{ij} is $rd(a_{ij}) * l_{comp}$ and $wr(a_{ij}) * l_{comm}$, where the parameter l_{comm} is the average delay multiplier incurred on the links of processors.

The execution of a task is divided into three phases (durations), the communication reading time, executing time, and communication writing time and these three stages cannot be interrupted. The ultimate optimization goal is to minimize the completion time of the entire application meeting constraints of hardware resources and precedence of tasks. If considering the cost of resources, balancing cost is another goal. The completion time, makespan, is only discussed here.

Implement constraint in ilog solver. ILOG Solver provides superior performance in constraint programming and can be embedded in C++ environment [7]. It is selected as the modelling platform to implement constraints and scheduling algorithm. The problem data is produced according to some probabilistic distribution, and then decision variables are selected to establish a constraint model of application. Finally, the problem is solved by a hybrid algorithm based on constraint programming and local search technology.

Problem data. The task graph parameters are stored in the data file and are represented as follows.

task_num and *arc_num* are the number of tasks and arcs.

pe_num is the number of processors.

$execu(n_i)$ is the execution time of each task.

$comp(n_i)$ is data size of each task.

$arc(i, j)$ is the precedence of tasks, and task j is a child node of task i.

$comm(n_i, n_j)$ is the size of communication data.

$rd(a_{ij})$ is reading time for communication.

$wr(a_{ij})$ is writing time for communication.

$pe\_capa(pe\_num)$ is the local storage capacity of processors.

l_{comp} and l_{comm} are the delay coefficients due to different storage space of data.

Decision variables and optimizing objective. In order to make use of the ILOG Solver algorithm to search the solution, a few decision variables are defined for the model. The scheduling algorithms mainly solve the resources distribution and determine the start time and end time of each phase. So the decision variables are selected as follows.

task_pe [*task_num*]: The processor tasks are executed.

arc_mem [*arc_num*]: The processor or share DRAM arc data are assigned.

task_mem [*task_num*]: The processor or share DRAM task data are allocated.

makespan: The completion time of the application and the optimization objective.

According to the application, each decision variable is given a domain of possible values.

Modelling the application. In ILOG Solver, there are rich constraints expresses. If the constraints are not depicted correctly, the best possible solution cannot be guaranteed. Here, the problem model must satisfy the resource constraints and precedence constraints. In resource allocating aspect, it includes data allocation to the memory and task distribution to a processor. Each communication or computation data must occupy a processor or share memory space. For reading communication data, if the communication data is assigned to the same processor as the consumer task, time is the shortest. For writing data, if the communication data and producer task are on the same processor, time is the shortest. Otherwise the reading time and the writing time are extended by multiplier l_{comm}. At the same time, the sum of data size of local storage is less than or equal to the memory capacity of the processor. It is assumed that share memory is sufficient. To any processor j, the capacity constraint is described as equation.

$$\sum comp(n_i) * (task\_mem[n_i] = j) + \\ + \sum comm(n_i, n_j) * (arc\_mem[arc(i, j)] = j) <= PE\_capa[j].$$

In scheduling aspect, all the tasks must satisfy the precedence constraints. After a task starts, its three phases cannot be interrupted. Before a task is executed, all the parent nodes data must be continuously read. When all reading communication queues have been obtained, a node can start to be executed immediately. After the task is completed, all writing communication of the children node must be written to the corresponding memory. When the task is assigned to a processor, the processor is taken during execution time. If a task has several parent tasks, these communication data must be read one after another. The writing communications have the same requirements. When a task is completed, the processor is released and next task starts. The three phases are defined by Class IloActivity and the time schedule is constrained by member functions startsAtStart, endsAtEnd, startsAtEnd and startsAfterEnd, etc. A task starts at the first reading communication and ends at the last writing communication.

Solving the problem based on Constraints programming and Local search. The allocating and scheduling are interdependent and cannot be isolated. The irrational allocation will lead to poor scheduling performance. Different strategies have been used to find a feasible solution satisfying the constraints in the past years. In this paper, constraints programming is implemented to solve the complex combinatorial problems in IBM® ILOG® Solver. When the problem model is established and the decision variables are defined, the search tree is created in search space of potential solution. Solver traverses the tree and finds a value for each decision variable while simultaneously satisfying the constraints and minimizing the objective. However, the order in which a variable is selected and the order in which domain value is tried can affect the quality of solution. So a hybrid algorithm is proposed to solve the problem and to guide the constraints programming using tabu search.

The search starts from the root node of the tree, meanwhile constraint propagation and backtrack are used to guide the search. The Ready task without parent nodes or whose parent nodes have been completed is collected in tasks queue and wait to be allocated. Here, the variable order is tackled through establishing the priority of tasks according to children number and the task with more children nodes has much chance to be picked out and is allocated to processor. The task with more children nodes prevent more tasks being executed, so these nodes will be allocated and scheduled firstly. At the same time, the ready task with least domain is also given priority when the solution space is explored. Then, the target processor is selected with lighter workload and less executing time. The computation data and communication data are also distributed to different storage spaces as possible as reducing the delay cost according to the pair of task-processor. Finally, the time of three phases is determined.

Solver finds a feasible solution by trying all possible combinations of values in the search space. If the search space is large, this approach is clearly time-consuming and ineffective. For this scheduling problem, it is difficult and unrealistic to find optimal solution in a short period. Considering the efficiency and complexity, heuristic information is introduced to speed the search process. Tabu search is local search and uses neighbour strategy to produce new solutions and is extensively applied in a scheduling field [9]. Tasks assigned to different processors are exchanged and the start time and the end time also are recom-

puted. This new solution must be feasible for constraints and the objective cost is evaluated. If the cost is better than that of the current solution and this new solution is accepted as a current one, otherwise current solution keeps unchanged. This algorithm continually improves the current solution until some stopping criterion is met. The criterion can be based on time, number of fail etc.

Local search is a greedy technique and is easy to be trapped in local minimum. In order to obtain a global optimal solution, the search selector IloMinimizeVar(env, cost, step) is used to guide the rest of the search process. In this problem, the variable 'cost' means makespan and 'step' represents decrement of current cost in the next loop. Here, the selector selects the leaf of the search tree which will minimize completion time of the application and stores the current best solution. The algorithm retrieves the original leaf and backtracks up the search tree in the absence of a better solution. When a better solution is found, the algorithm adds a constraint that objective cost must be less than or equal to the current cost minus 'step'. In selector IloMinimizeVar(env, cost, step), the new constraint acts as an upper bound of completion time and the algorithm employs reverse binding of the current best solution. The upper bound takes the place of the domain value of the last node and algorithm implements constraints propagation of tasks. If the constraints are met, it reactivates the value of task nodes by a backtracking method and stores the new best solution. The process is repeated until there is no better solution.

Simulating results. ***Benchmark graphs with different features.*** The proposed scheduling algorithm is implemented on benchmark sets and each benchmark instance represents a task graph. A task graph must be able to reflect the general characteristics of an application. Considering the diversity of applications, several common features are discussed. In these instances, the number of tasks and communication arcs are important factors and CCR is used to evaluate the ratio of communication time to computation time [8]. Meanwhile, data-intensity or computation-intensity is also considered, which means data storage capacity or execution time has a greater proportion in the scheduling process of applications. Sometimes task executing time is descried as coarse or fine based on task granularity. These several parameters interrelate with each other from different aspects, for example, both of granularity and CCR involve the ratio of communication and computation. The computation and communication latency are also considered and three situations are discussed here. Three cases are: no latency, only computation latency, both communication latency and computation latency respectively. The computing platform consists of several processors with different storage memory and storage capacity is considered.

The searching process of the proposed algorithm. This proposed algorithm uses a hybrid algorithm, which involves the constraint programming and tabu search (denoted as CP + TB). The algorithm is simulated in the platform with six processors and Fig. 1 displays the makespan change of a random task graph with 20 tasks during the search progress. The search is stopped by time limit and the process demonstrates that the algorithm could gradually find a better solution. More results need further work by comparing with other algorithms and testing on the different types of data.

Results with different CCR. In this example, 300 tasks with different CCR are discussed and no latency is considered. The executing cost and communicating cost are produced randomly and applications are data-intensity. A pure CP algorithm is implemented in the same platform to test the proposed algorithm and this algorithm uses the constraint propagation strategy and backtracking method [3]. The algorithm restarts the search process until the time meets the requirement. The pure CP tries all possible values for constraint variable and finally finds the optimal solution. But it is less efficient to deal with large scale applications. The results are depicted in Fig. 2 and the hybrid algorithm always demonstrates better performance than the pure CP. Tabu search can speed the convergence, but it is often trapped into the local optimal solution. The proposed algorithm combines the global search capacity of the pure CP and the speed superiority of TB, so the algorithm can find better results in a limited time period. Because the cost is random, scheduling length does not always increase with the increase in the density.

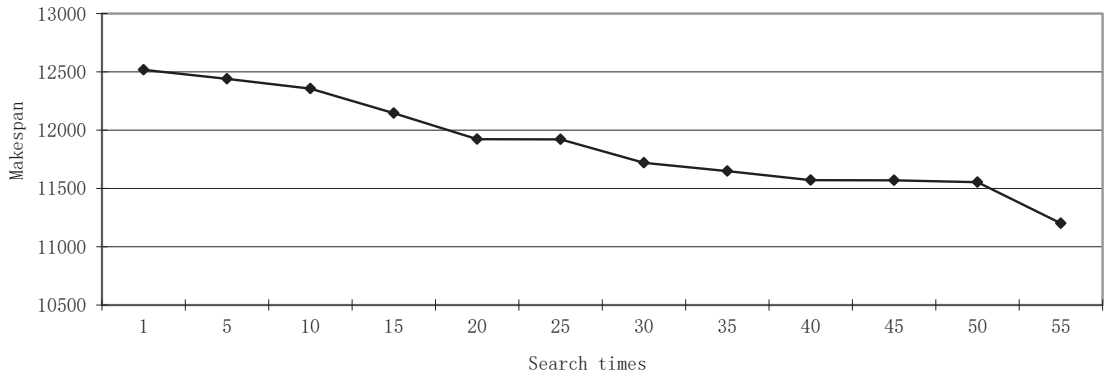

Fig. 1. The makespan change of a random task graph

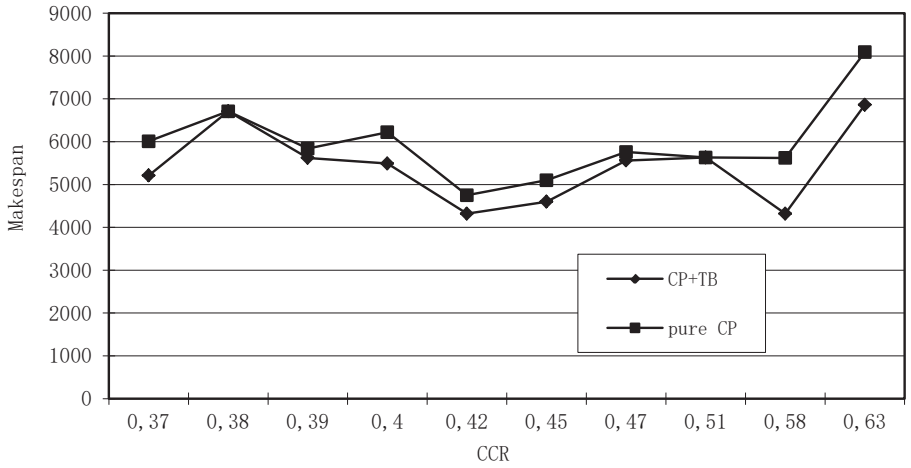

Fig. 2. The makespan of tasks with different CCR

Results with different time latency. When the data is distributed in different memory, the time delay has great influence on the makespan. The scheduling results of two algorithms are given in Fig. 3, 4. Only the communication delay is considered in Fig. 3 and both computation and communication delay are taken into account in Fig. 4. It is concluded that the proposed algorithm explores more extensive solution space and has lager probability to find a better solution in the same time. The advantage is obvious with the increase of the task scale.

Here, cost of tasks and communications are according to position distribution of [1000, 5000]. With the increase of tasks, applications have longer com-

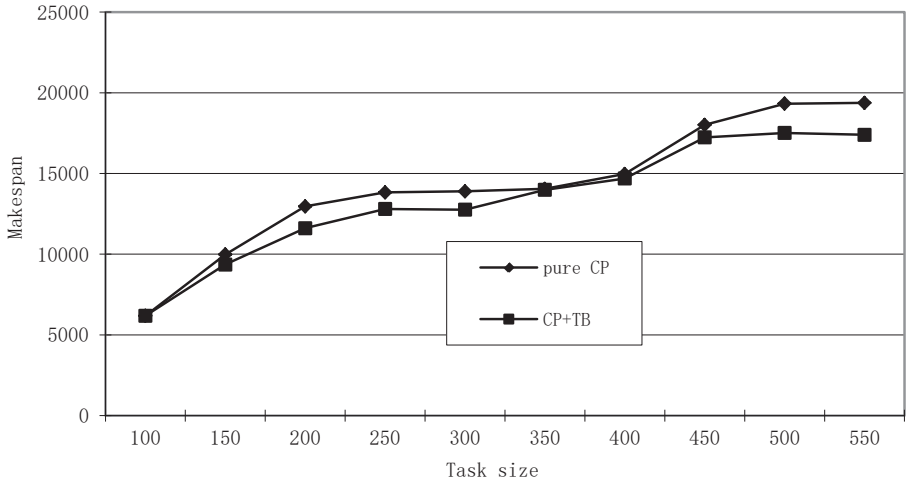

Fig. 3. The makespan of tasks with communication latency

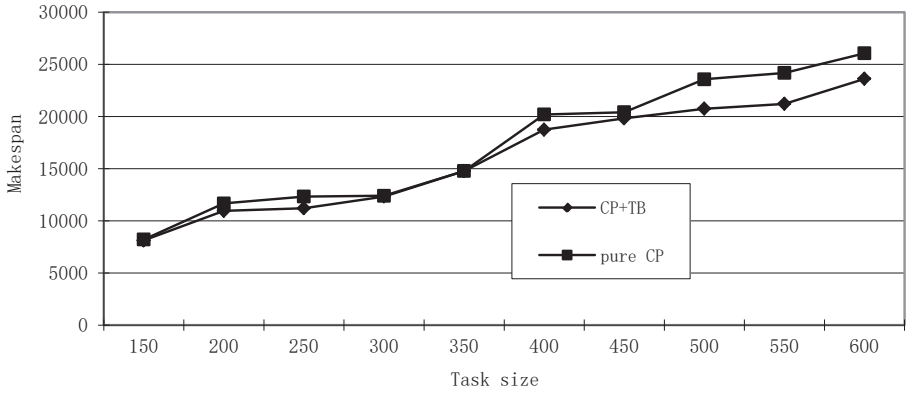

Fig. 4. The makespan of tasks with communication latency and computation latency

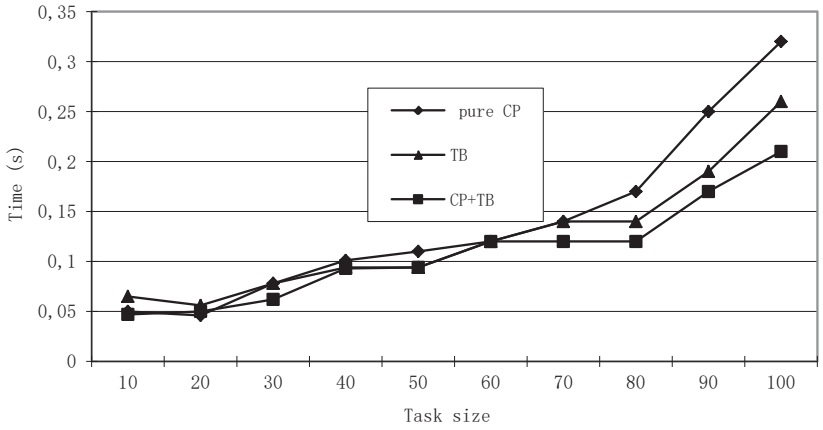

Fig. 5. The time of finding optimum solution

pletion time and the hybrid algorithm reflects advantage. Satisfying the constraints, the pure CP always tries every optional variable value and relies heavily on mathematical models. During the search process, Tabu search uses the information of the original solution and always improves the current solution. When task sizes are bigger, the proposed algorithm can find a rational solution in a shorter period. So this algorithm is suitable especially for the large-scale complicated problems that traditional methods cannot solve well.

Comparison of algorithm time. In the following task instances, the optimal solution is evaluated based on the length of the critical path through traversing the graph structure. As shown in Fig. 5, three algorithms are compared regarding the time to find an optimal solution and the algorithms are pure CP, TB and the proposed algorithm. When the task sizes are smaller, the times are almost same. With the increment of tasks, the hybrid algorithm manifests the superiority and is much faster to search the optimal solution.

Conclusions. Sharing a variety of available resources, the multiprocessors system is a promising approach to improve the computational capacity in the parallel system. However, due to the complexity of the scheduling problem, it is hard that different architectural attributes, such as message routing strategy, computation, and processor heterogeneity, are orchestrated to perform an application. Otherwise tasks can fail to exploit the true potential of multiprocessor systems and the gains from parallelism can be offset. This research aims to propose a hybrid algorithm to solve the resource-constrained scheduling based on constraints programming and local search. At the same time, tasks with different features are also investigated to find how the performances of applications are influenced in a specific target platform. The proposed algorithm for the multiprocessor scheduling problem is an effective strategy and the technique can improve search efficiency and solution performance for practical problems.

Acknowledgements. This work was supported by Research Fund for Scientific and Technological project of Henan Province No.142102210112.

References/Список літератури

1. Davis, R.I. and Burns, A., 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, Vol. 43, No. 4, pp. 1–44.

2. IBM Corporation, 2010. Cell Broadband Engine Programming Handbook, *http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs9F820A5FFA3ECE8C8725716A0062585F*.

3. Benini, L., Lombardi, M., Milano, M. and Ruggiero, M., 2008. Optimal resource allocation and scheduling for the CELL BE platform. *Annals of Operations Research*, Vol. 184, No. 1, pp. 51–77.

4. Leila, I. and Driss, G., 2011. Performance Evaluation of Convolution on the Cell Broadband Engine Processor. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 2, pp. 337–351.

5. Edis, E.B. and Oguz, C., 2012. Parallel machine scheduling with flexible resources. *Computers & Industrial Engineering*, Vol. 63, No. 2, pp. 433–447.

6. Heinz, S., Ku, W.Y. and Beck, J. C., 2013. Recent Improvements Using Constraint Integer Programming for Resource Allocation and Scheduling. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. 10th International Conference*, CPAIOR 2013, Yorktown Heights, NY, USA, May 18–22, 2013.

7. IBM Corporation, 2011. *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual v12.4.*

8. Canon, L.C. and Jeannot, E., 2009. *Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules. Research Report*. Research Report RR-6895, INRIA.

9. Thevenin, S., Zufferey, N. and Widmer, M., 2013. Tabu search for a single machine scheduling problem with discretely controllable release dates. In: Slovenian Society Informatika, *12th International Symposium on Operational Research SOR'13 in Slovenia*. Dolenjske Toplice, September 25–27, 2013. Slovenian Society Informatika – Section for Operational Research, Ljubljana, Slovenia.

Мета. Декілька процесорів можуть бути об'єднані разом у багатоядерній платформі й мати справу з багатомасштабними науковими пробле-

мами із-за надлишку обчислень і обширного розпаралелювання. Ця робота, в основному, направлена на вирішення проблеми багатозадачного планування на цій платформі зі спільною пам'яттю й каналом зв'язку.

Методика. Як правило, інтенсивне обчислення завдання порціонується у крупні підзадачі так, що завдання виконуються паралельно (одночасно) для підвищення продуктивності обчислень. Запропонований алгоритм використовує програмування в обмеженнях для знаходження відповідного рішення та використовує локальний пошук для покращення виявлення рішення й прискорення процесу.

Результати. Спільне використання різних наявних ресурсів, багатопроцесорне планерування є складним комбінаторним завданням оптимізації та проблемою обмежених ресурсів. Досліджено, як отримати раціональне рішення або субоптимальне рішення в короткий період часу. У той же час, досліджені завдання з різними характеристиками, аби знайти, як характеристики додатків впливають на питому мету платформи.

Наукова новизна. Коли завдання призначені різним процесорам, мають бути розглянуті різні обмеження ресурсів, у тому числі зберігання даних, обчислювальна вартість, пріоритетність завдань і вартість зв'язків між процесорами. Розглядається гібридний алгоритм, заснований на пошуку із заборонами, з метою оптимізації часу завершення додатків, задовольняючи вказаним обмеженням.

Практична значимість. Алгоритм реалізований у середовищі IBM ILOG CPLEX Optimization Studio environment і дає кращі результати в порівнянні з іншими алгоритмами. Пропонований алгоритм є ефективною стратегією, і ця методика дозволяє підвищити ефективність пошуку й продуктивність для вирішення проблеми багатопроцесорного планування.

Ключові слова: *багатопроцесорність, планування завдань, програмування в обмеженнях, пошук із заборонами*

Цель. Множественные процессоры могут быть объединены вместе в многоядерной платформе и иметь дело с многомасштабными научными проблемами из-за избытка вычислений и обширного распараллеливания. Эта работа, в основном, направлена на решение проблемы многозадачного планирования на этой платформе с общей памятью и каналом связи.

Методика. Как правило, интенсивное вычисление задания порционируется в крупные подзадачи так, что задачи выполняются параллельно (одновременно) для повышения производительности вычислений. Предложенный алгоритм использует программирование в ограничениях для нахождения подходящего решения и использует локальный поиск для улучшения обнаружения решения и ускорения процесса.

Результаты. Совместное использование различных имеющихся ресурсов, многопроцессорное планирование является сложной комбинаторной задачей оптимизации и проблемой ограниченных ресурсов. Исследовано, как получить рациональное решение или субоптимальное решение в короткий период времени. В то же время, исследованы задачи с различными характеристиками, чтобы найти, как характеристики приложений влияют на удельную цель платформы.

Научная новизна. Когда задачи назначены разным процессорам, должны быть рассмотрены различные ограничения ресурсов, в том числе хранение данных, вычислительная стоимость, приоритетность задач и стоимость связей между процессорами. Рассматривается гибридный алгоритм, основанный на поиске с запретами, с целью оптимизации времени завершения приложений, удовлетворяя указанным ограничениям.

Практическая значимость. Алгоритм реализован в среде IBM ILOG CPLEX Optimization Studio environment и дает лучшие результаты по сравнению с другими алгоритмами. Предлагаемый алгоритм является эффективной стратегией, и эта методика позволяет повысить эффективность поиска и производительность для решения проблемы многопроцессорного планирования.

Ключевые слова: *многопроцессорность, планирование задач, программирование в ограничениях, поиск с запретами*

Рекомендовано до публікації докт. техн. наук В. В. Гнатушенком. Дата надходження рукопису 28.07.15.