The research and implementation of The Critical Path on a Processor (CPOP) Algorithm based on Pi calculus

He Yuan [a], Yongzhe Zhao [b] and Hui Kang [c, *]

College of Computer Science and Technology, Jilin University, Changchun 130012, China

[a]yhsweetlife@163.com, [b]yongzhe@jlu.edu.cn, [c]kanghui@jlu.edu.cn

Abstract. Task scheduling algorithms in heterogeneous computing environment often provide limited performance owing to their low efficiency. This paper models and implements the first two phases of the Critical Path (CP) on a Processor Algorithm with the functional programming paradigm. Firstly, an overall research is conducted on the CPOP algorithms, then the logical relationship of the first two phrases of the algorithm is modeled based on Pi calculus theories, and finally the algorithm is implemented by nPict programming language. The experimental results demonstrate that the implementation efficiency of the new programming algorithm is higher than that of the traditional C++ language. Therefore, Pi calculus can be applied to the three phases of the CPOP algorithm to improve the task scheduling efficiency. This paper aims to prove that the CPOP algorithm can be more efficient than ever by using Pi calculus, through modeling and comparing the topological structures of the different types of task scheduling models.

Keywords: Heterogeneous system; Task scheduling algorithm; Pi calculus; nPict.

1. Introduction

Currently, the heterogeneous computing system plays a more and more important role in dealing with complex problems in both industrial production area and ecommerce area. Heterogeneous computer clusters, which support the execution of parallel applications, are implemented in heterogeneous systems to complete the task quickly. The efficiency and the load balancing of the running parallel applications depend on the way in which the tasks are scheduled onto the processors. However, the task scheduling problem is an NP-complete problem, so it is impossible to be accomplished in polynomial time. Therefore, the researchers are studying how to make the scheduling algorithm more efficient and effective. Now, the task scheduling algorithm can be divided into heuristic method and random search, and the former in a heterogeneous system can be further divided into list scheduling Heuristics (HEFT) [2], clustering heuristics and task Duplication Heuristics. The CPOP algorithm [2] is a representative heuristic algorithm. This paper focus on the realization of the algorithm and the assignment relationships understanding among tasks.

This paper studies on the task scheduling algorithm in heterogeneous system, then models the first two phases of the CPOP algorithm using the Pi calculus theory, and finally implements the algorithm with nPict language. By carrying out an experimental comparison between traditional language and new parallel programming language, it turned out that the distributed concurrent Pi calculation and the nPict language are more efficient than the traditional high-level language during processing distributed tasks. Furthermore, it would be interesting to explore whether this implementation method can be extended to the last phase of the CPOP algorithm in a heterogeneous system.

1.1 Dag.

the parallel application is represented by a directed acyclic graph, or DAG [12], defined by the tuple (T, E), {T = {t_i | i ∈ [1, n]}, and it is a set of all the nodes in DAG, each node representing each task of the distributed application. E = {e_i | $i \in [1,m]$, $e_i = (t_s, t_e)$, $t_e \in$ succ(t_s)}, and each edge $e_i = (t_s, t_e)$ representing a precedence constraint and a communication message between two tasks.

Definition 1.1 The recursive definition of upward rank of a task is $rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)}(\overline{c_{i,j}} + rank_u(n_j))$. The recursive definition of node downward rank of a task is $\text{rank}_d(n_i) = \max_{n_j \in pred(n_i)}\{rank_d(n_j) + \overline{w_j} + \overline{c_{j,i}}\}$.

The upward rank value of exit nodes is equal to $\overline{w_{exit}}$. $\overline{w_i}$ represents the average computation cost of task n_i. $\overline{c_{i,j}}$ represents the average communication cost from task n_i to task n_j. $succ(n_i)$

represents the set of immediate successors of task n_i. $pred(n_i)$ represents the set of immediate predecessors of task n_i.

1.2 The CPOP Algorithm.

As a typical algorithm of list scheduling heuristics, the CPOP algorithm consists of three parts. The first part is the task prioritizing phase, in which upward rank and downward rank values for every task in DAG are computed and the sum of them will be the priority value of every task. Then the scheduling queue will be drawn up in order of decreasing priority. The second part is to select the CP processor. The CP is defined as the path from an entry task to an exit task for which the sum of the computation costs of tasks and the communication costs of edges is maximal. The communication costs on the same processor are too small to be ignored. So the sum of computation costs of the tasks located on the CP determines the lower bound of the final schedule length. The processor of the CP is the one that minimizes the cumulative computation costs of the tasks on the CP. The third part is to deal with nodes not located on CP considering an insertion-based scheduling policy. The selected task on the CP is scheduled on the processor of the CP. The rest of tasks is assigned to a processor which ensures the shortest execution finish time of task nodes.

The process of the CPOP algorithm is defined as follows: (the first two phases)

(1) First, set the computation time of every node in DAG and the communication time of tasks.

(2) Compute upward rank value ($Rank_u$) of the nodes.

(3) Compute downward rank value ($Rank_d$) of the nodes.

(4) Compute the priority list of the nodes, which is equal to $Rank_u$ + $Rank_d$.

(5) Find out the children nodes whose priority are equal to that of the CP value from entry node.

(6) Select the processor which minimizes the earliest finish time.

2. Preliminaries and computational model

2.1 Pi Calculus.

Pi calculus [9] is proposed by Professor Milner, a Turing Award Laureate. This theory promotes the communication process calculus and allows to transfer the name of a channel in communication, enabling the Pi calculus to describe the runtime changes of the communication topological structure. Thus Pi calculus has strong ability of expression and inherits the concise semantic theory of CCS, bisimulation. Therefore, it has been used in the design of programming language and the analysis and verification of distributed systems. This paper uses programming with lists of Pi calculus [13].The nodes in the DAG are signified by the elements of the list in the modeling.

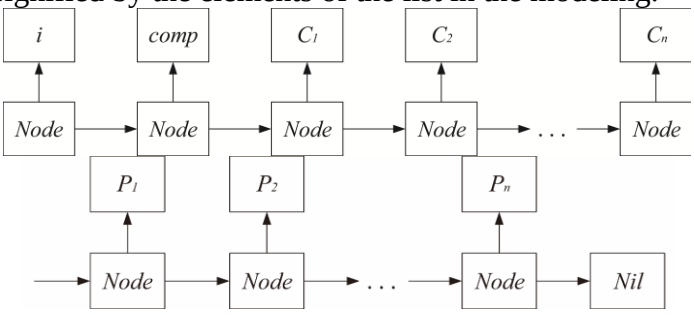

Fig.1 DAG node list structure

i : The location of task; $P_1, P_2 \ldots, P_n$: The list of the parent task; $C_1, C_2 \ldots, C_n$: The list of the child task; *comp*: The computation cost of task.

Parent task and child task use the same structure of list as shown in Fig.2. The list structure of the priority queue is showed in Fig.3.

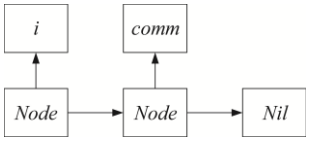

Fig.2 The list structure of node C_i and P_i (*i:* The location of task; *comm:* The communication cost of the task)

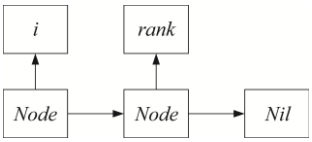

Fig.3 The list structure of priority queue (*i:* The location of task; *Rank:* The priority value of the task)

2.2 Modeling.

The following is the description of system behaviors: $System \triangleq^{def} Step_1.Step_2.Step_3.Step_4$

$Step_1$ Computing $Rank_d$

$$P \triangleq^{def} \overline{send} < 0, r, d > |Q \quad (1)$$

$$Q \triangleq^{def} send(m,t,d).\bar{t} < nc >.c(comp,l).\overline{send1} < (comp+m), l, d > |S|Cons(V,L) < t > \quad (2)$$

$$S \triangleq^{def} send1(r,l,d).\bar{l} < nc >.c(x,l').\bar{x} < nc > c(comm,l'').\bar{d} < n'c' >.c'(x',d').\overline{d'} < n'c' > c'(drank,\ d'').\overline{send2} < (r+comm), drank, l'', d'' > |T|Cons(V,L) < l > |Cons(V,L) < x > |Cons(V,L) < d > |Cons(V,L) < d' > \quad (3)$$

$$T \triangleq^{def} send2(rank, drank, l, d).(if(rank > drank) \Rightarrow (drank = rank).\overline{send3} < drank, l, d > |R) \quad (4)$$

$$R \triangleq^{def} send3(r,l,d).(Case\ \ l\ \ of\ \ Nil? \Rightarrow \overline{DOWN} < d > Else \Rightarrow\ \overline{send} < r, l, d > |Q) \quad (5)$$

$Step_2$ Computing $Rank_u$

$P \triangleq^{def} \overline{recv} < 0, r, u > |Q$

$Q \triangleq^{def} recv(m,t,u).\bar{t} < nc >.c(comp,l).\overline{recv1} < (comp+m), l, u > |S)|Cons(V,L) < t >$

$S \triangleq^{def} recv1(r,l,u).\bar{l} < nc >.c(x,l').\bar{x} < nc > c(comm,l'').\bar{l''} < nc > c(comp,l''').\bar{u} < n'c' >.c'(x',u').c'(x',u').\overline{u'} < n'c' > c'(urank,u'').\overline{recv2} < (r+comm+comp), urank, l'', u'' >.T|Cons(V,L) < l > |Cons(V,L) < x > |Cons(V,L) < l'' > |Cons(V,L) < d > Cons(V,L) < d' >$

$T \triangleq^{def} recv2(rank, urank, l, u).$ if(rank > urank) $\Rightarrow (urank = rank).\overline{recv3} < urank, l, u > |R)$

$R \triangleq^{def} recv3(r,l,u).(Case\ \ l\ \ of\ Nil?\ \Rightarrow \overline{UP} < u > Else\ \Rightarrow\ \overline{recv} < r, l, u > |Q)$

$Step_3$ Computing $Rank$

$P \triangleq^{def} DOWN(d).UP(u).\overline{getRank} < d, u, r > |R$

$R \triangleq^{def} getRank(d,u,r).\bar{d} < nc >.c(x,d').\bar{u} < nc >.c(y,u').\bar{r} < nc >.c(z,r').(z = (x+y)).\overline{getRank} < d', u', r' > |R|Cons(V,L) < d > |Cons(V,L) < u > |Cons(V,L) < r >$

$Step_4$ Selecting the CP processor

$P \triangleq^{def} \overline{findCP} < rank, l, r > |R$

$R \triangleq^{def} findCP(rank,l,r).\bar{l} < nc >.c(x,l'.\bar{r} < nc >.c(y,r').\bar{y} < nc >.c(rank',r'').\overline{findCP1} < rank, rank', l', r'' > |S|Cons(V,L) < l > |Cons(V,L) < r > |Cons(V,L) < y >$

$S \triangleq^{def} findCP1(rank, rank', l, r).if(rank = rank') \Rightarrow \bar{l} < nc >.c(x,l').\overline{CP} < x > |T|Cons(V,L) < l > |\overline{findCP2} < rank, l', r > |W$

$T \triangleq^{def} CP(x).\tau$

$W \triangleq^{def} findCP2(rank,l,r).(case\ l\ of\ Nil? \Longrightarrow \varepsilon\ \ Else \Longrightarrow \overline{findCP} < rank, l, r > |R).$

Then this paper takes $Step_1$ as an example and introduce the whole modeling process of the list programming. Pi calculus theory models based on process channel figure, which shows the real spatial displacement in computing task nodes. Figure 4 is the process channel figure of $Step_1$ and the modeling of $Step_2$ $Step_3$ and $Step_4$ is similar to it.

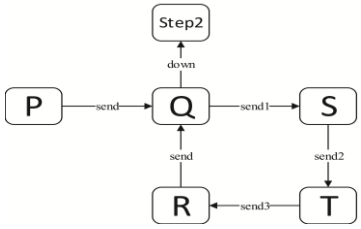

Fig.4 Channel figure between progresses

Formula (1): Process P sends the beginning time, downward rank list and the children nodes list of the entry node to Process Q through the channel send. Formula (2): Process Q sends the sum of the computing cost of the present node and the received period to Process S through channel send1. Formula (3): Process S works out result (r + comm) by adding the communication cost to the received period and sends it to Process T with the drank value taken from the downward rank list through channel send2. Formula (4): Process T compares the digits of rank and drank. If the former is bigger, the value of the downward rank list will be updated, and the new drank list will be sent to Process R through channel send3. Formula (5): Process R checks if the task node list (l) is empty. If so, it means that ergodic process is finished, and the updated drank list will be sent to $Step_2$ through channel down. And if not, it will be executed recursively to Process Q.

3. Implementation of CPOP algorithm

3.1 The First phase.

```
Procedure DownwardList(ln,ld,i)

Input:ln is a list of n tasks, ld is a null downward rank list of n tasks, i is the location of the current task, count is the number of tasks.
Output:ld is a downward rank list after updated.
BEGIN
    for i = 1 to count do
        if i == count then
            send ld though channel DOWN
        else
            take the structure of current task from ln.
            take the downward rank value of current task from ld.
            update the list ld of children tasks
END
```

```
Procedure UpwardList(ln,lu,i)

Input:ln is a list of n tasks, lu is a null upward rank list of n tasks,i is the location of the current task count is the number of tasks
Output:lu is a upward rank list updated.
BEGIN
    for i = count to 1 do
        if i == 1 then
            send lu though channel UP
        else
            take the structure of current task from ln.
            take the upward rank value of current task from lu.
            update the list lu of parent tasks
END
```

Compute Downward Rank List (ld):

(1) According to node position i, taking the element structure from the nodes list ln and the earliest completion time of this node from the downward rank list ld, send the sum of computation cost of present node and drank value to children nodes list of node i. If children nodes list is not empty, continue the third step. If not, skip to the fourth step.

(2) If the value the child node received is bigger than its drank value, then update ld list and recursively call second step.

(3) Print List ld.

Compute Upward Rank List (lu):

(1) According to position i, taking the structure of this node, and then taking the earliest completing time of this node, finally send computation cost of present node to node i's father node list, if father node list is not empty, enter the third step; if not, skip to the forth step.

(2) Choosing maximum sum value of arrival time of father node, communication cost of children node and computation cost of father node to update to lu list.

(3) Print List lu.

3.2 The Second phase: select processor.

```
Procedure Selected(ln,lr)

Input:ln is a list of n tasks, lr is a priority rank list of n tasks
Output:pid is the serial number of the CP processor
BEGIN
    Let the initial task n be entry task and let |CP| be priority value of entry task
    while task n is not exit task do
        Select the task from children tasks where priority value of task = |CP|
        Let the CP task list contain the selected task
        Let task n be the selected task
    endwhile
    Select the CP processor which minimizes finished time of the CP task list
END
```

(1) Initial state: Starting from entry node, to search the child node, which the node priority is equal to that of parent node. Once found, skip to second step; if children nodes list is empty, algorithm ends.

(2) Print the serial number of the CP processor.

4. Experiment and analysis

Allowing for the complicated topology of task DAG figure, this paper respectively imitates five conditions of 10,20,30,40 and 50 nodes. Each case simulates two topologies of depth-first and width-first DAG figure to do comparison. The topology of depth-first means the depth of DAG is larger than width of DAG. Width-first is a type of experimental comparison based on DAG which shows the width is much greater than depth. Depth-first means the depth is much greater than width.

In the centos 6.5 operating system of Linux platform, with memory of 1g and hard disk of 20g, an experiment on the nPict and C++ implementation of the CPOP algorithm is carried out. Referring to the Linux system, real time is wall clock time of process from start to finish of the call, including time slices used by other processes and time the process spends blocked (waiting time of entering IO). User time is the amount of CPU time spent in the user mode (outside the kernel) within the process, which is only actual CPU time of executing the process. Sys time is the CPU time spent in the kernel within the process. It means executing CPU time spent in system call within the kernel.

4.1 Width-first with horizontal comparison.

Fig.5 and Table 1 are the real time comparison after experimenting on five conditions. As shown in Fig.5, under circumstances of the same number of nodes, the time nPict cost is far less than that of C++, with the average difference of more than 200ms. Allowing for the extra time spent on operating system process call, nPict is still superior to C++. With increase of nodes, the time advantage remains great. Thus when the number of nodes is more than 50, nPict's efficiency is much higher than C++.

Table 1 Width-first real time comparison (ms)

Number of nodes	C++	nPict
10	365.8	92.4
20	373.6	105
30	389.1	124.1
40	397.1	133.8
50	400.0	148.1

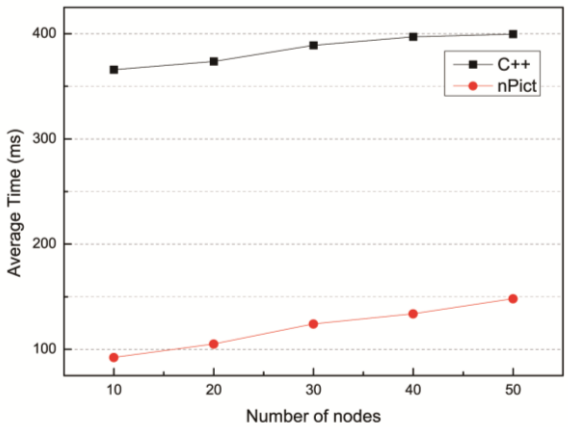

Fig.5 Width-first real time comparison (ms)

Figure 6 and Table 2 are user time contrast of the CPOP algorithm experiment in breadth-first case. As can be seen from the figure, nPict language is still more efficient than C++ language in user time. User time is actual CPU time spent by executing the program, excluding blocked time of process.

Table 2 Width-first user time comparison (ms)

Number of nodes	C++	nPict
10	306.6	84.3
20	307.2	96.4
30	305.5	115.5
40	310.8	125.3
50	309.6	138.3

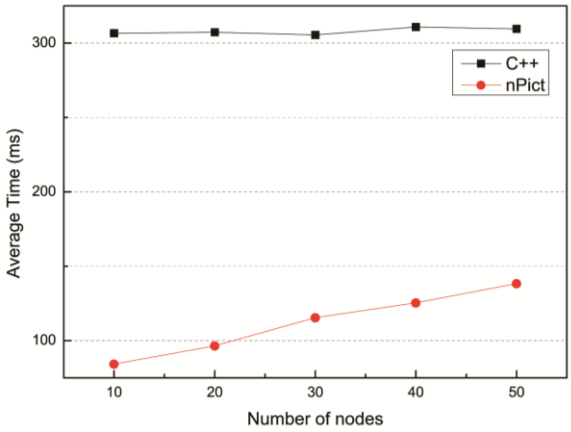

Fig.6 Width-first user time comparison (ms)

Figure 7 and Table 3 are sys time contrast of the CPOP algorithm experiment in breadth-first case. As a whole, sys time of nPict language occurrence trend is steady. In the meantime, the trend of C++ presents shock and irregular. That is because sys time is the time spent by system call in the kernel. System call in nPict has outstanding performance.

Table 3 Width-first sys time comparison (ms)

Number of nodes	C++	nPict
10	31.9	2.9
20	32.4	3
30	36.8	2.6
40	33.2	3
50	32.3	3

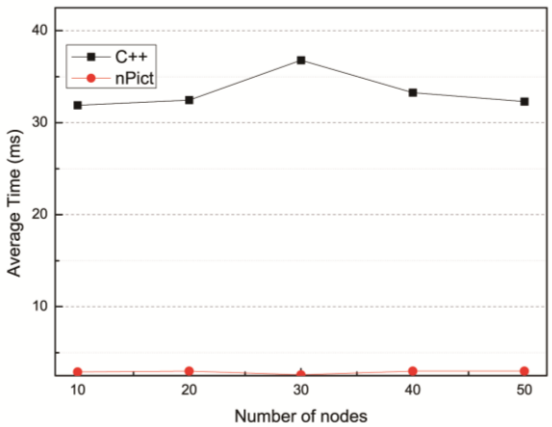

Fig.7 Width-first sys time comparison (ms)

4.2 Depth-first with longitudinal comparison.

Fig.8 and Table 4 shows, in case of depth-first, the longitudinal comparison of the real time that implements the CPOP algorithm using two languages. Allowing for that real time is the actual time of operating system process call, it includes the blocked state of other processes. So real time is longer than user time and sys time. According to Fig 4.4, we can conclude that the efficiency of using nPict to implement the algorithm is much better than using C++ when having the same number of nodes. In addition, with the number of nodes increasing, the advantage in time still won't decrease.

From perspective of time increase, the time interval of longitudinal depth-first is a little bit smaller than that of horizontal width-first, and the general trend of longitudinal depth-first is more stable.

Thus it can be possible to conclude that when the number of nodes is above 50, the advantage of this programming method this paper introduced in time is still in presence.

Table 4 Depth-first real time comparison (ms)

Number of nodes	C++	nPict
10	384.7	96.8
20	391.2	104.6
30	398.3	129.3
40	399.2	135.2
50	408.9	149.4

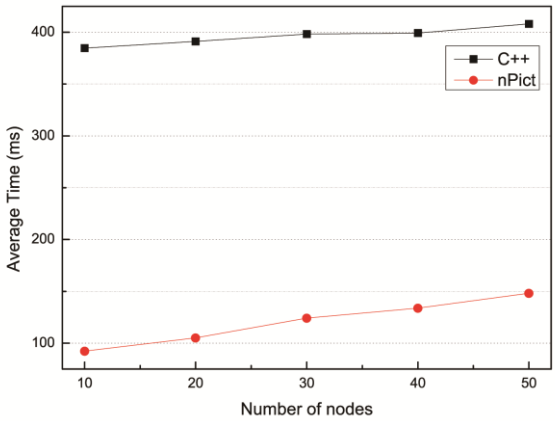

Fig.8 Depth-first real time comparison (ms)

Fig.9 and Table 5 shows, in the case of depth-first, the longitudinal comparison on the user time that implements the CPOP algorithm in two languages. According to Fig 4.5, the average consumption of nPict is above 200ms less than that of C++.

Table 5 Depth-first user time comparison (ms)

Number of nodes	C++	nPict
10	304.9	86.2
20	312.8	95.6
30	323.8	119.5
40	317.4	125.6
50	329.9	140.1

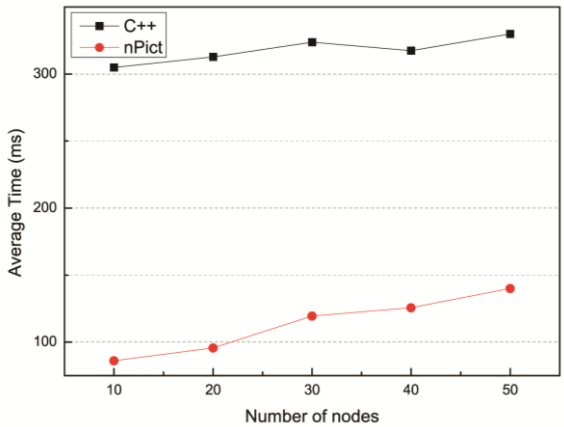

Fig.9 Depth-first user time comparison (ms)

Figure 10 and table 6 shows, in the case of depth-first, longitudinal comparison of the sys time that implements CPOP algorithm in two languages. From perspective of the kernel mode time, nPict still has the greater advantage. However, considering the changing curve of width-first sys time, in the case of depth-first, with the increase of node, sys time becomes short and short, which indicates that the time spent on system call is shorter. It indirectly suggests that the CPOP algorithm implemented by nPict can be better implements at the depth-first DAG.

Table 6 Depth-first Sys Time Comparison (ms)

Number of nodes	C++	nPict
10	41.3	5.5
20	36.5	3.8
30	36.7	3.9
40	35.1	3.8
50	33.4	3

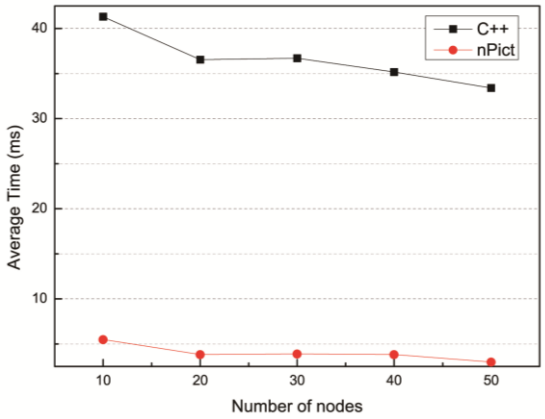

Fig.10 Depth-first Sys Time Comparison (ms)

5. Conclusion

This paper firstly makes research on the task scheduling algorithm under heterogeneous system, then exploits the Pi calculus theory to do modeling of first two phases of the CPOP algorithm, finally implements with nPict language. By carrying out an experimental comparison between traditional language and new parallel programming language, it turned out that distributed concurrent Pi calculation and nPict language are more efficient than traditional high-level language in dispatching and processing distributed tasks. It would also be interesting to see whether this implementation method can extend to the last phase of the CPOP algorithm in a heterogeneous system.

References

[1]. Sewell P, Wojciechowski P T, Unyapoth A. Nomadic pict: Programming languages, communication infrastructure overlays, and semantics for mobile computation. [J]. Acm Transactions on Programming Languages & Systems, 2010, 32 (4): 163-172.

[2]. Topcuouglu H, Hariri S, Wu M Y. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing [J]. IEEE Transactions on Parallel & Distributed Systems, 2002, 13 (3): 260-274.

[3]. Kang Y, Zhang D. A Hybrid Genetic Scheduling Algorithm to Heterogeneous Distributed System [J]. Applied Mathematics, 2012, 03 (7): 750-754.

[4]. Pawel T. Wojciechowski.: Nomadic Pict Language Libraries Release 1.0-alpha December 18, 2000.

[5]. Pawel T. Wojciechowski.: The Nomadic Pict System Release 1.0-alpha Documentation and user's manual. December 19, 2000.

[6]. Wu M Y, Gajski D D. Hypertool: A Programming Aid for Message-Passing Systems [J]. Parallel & Distributed Systems IEEE Transactions on, 1990, 1 (3): 330-343.

[7]. Pierce B C. Concurrent objects in a process calculus [C] Proceedings of the International Workshop on Theory and Practice of Parallel Programming. Springer-Verlag, 1994: 187-215.

[8]. Pierce B C, Turner D N. Pict: A Programming Language Based on the Pi-Calculus [C] Proof, Language & Interaction: Essays in Honour of Robin Milner. MIT Press, 2000: 455 - 494.

[9]. Milner, Robin. Functions as Processes. In Research Report 1154, INRIA, Sophia An- tipolis. Final version." in J. Mathem. Struct. In Computer Science. (1990)

[10]. KANG Hui, ZHANG Shuang-shuang, MEI Fang.: Petri net translation of recursion π-calculus [J].Journal of Jilin University Engineering and Technology Edition, 2014, 44 (1): 142-148.
[11]. Reakook Hwang, Mitsuo Gen, Hiroshi Katayama.: Comparison of multiprocessor task scheduling algorithms with communication costs. ScienceDirect. (2008)
[12]. Stavrinides G L, Karatza H D. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations [J]. Journal of Systems & Software, 2010, 83 (6): 1004-1014.
[13]. Milner R. Communicating and mobile systems - the Pi-calculus. [J]. Cambridge University Press New York, 1999, 42 (2-3): 100-191.
[14]. Wojciechowski P T, Sewell P. Nomadic Pict: Language and Infrastructure Design for Mobile Agents [C] International Symposium on Agent Systems and Applications, 1999 and Third International Symposium on Mobile Agents. Proceedings. IEEE, 1999: 2-12.
[15]. Unyapoth A, Sewell P. Nomadic pict: correct communication infrastructure for mobile computation [J]. Acm Sigplan Notices, 2001, 36 (3): 116 - 127.