Resource-aware fluid scheduling with time constraints for clustered many-core architectures

Anqi Yin¹, Yuanbo Guo¹* and Di Tang¹

¹ Department, Institute of Information Science and Technology, Zhengzhou, Henan, 450001, China

*Corresponding author’s e-mail: dubito@non-you.com

Abstract. With the rise of many-core processors, real-time applications processing obtains better performance acceleration and parallelization freedom. A validate schedule is able to minimize system resource consumption under the time constraints, which has become a focus recently. This paper establishes an accurate fluid task model and an optimal integer linear programming (ILP) model further. These models not only realize the flexible parallel scheduling, but also achieve co-scheduling between computing and communication resources. Moreover, a heuristic algorithm is presented to generate near-optimal solutions within polynomial time. The experimental results show that the proposed approach is validate and capable to balance the time and resource consumption effectively. It leads the improvement of schedulability and validity up to 68.67% and 24.67% respectively, with 13.16% reduction of the system density averagely.

1. Introduction

Many-core processors are high-performance computing systems that integrate parallel computing and communication resources [1]. Clustered many-core platforms whose computing granularity are getting smaller is a network constituted of multi-core clusters [7]. Such platform provides the possibility to parallelize a task in different way, which is called as parallelization freedom [9].

Parallel scheduling is an efficient way to improve performance, but it does not bring benefits all the time due to the additional overhead caused by parallelization [2]. Excessive parallelization will make the total computation amount become too large and result in an overlarge system “density”. The clustered many-core architecture extends the flexibility of tradeoff between density and time. But it also greatly increases the scheduling difficulty because the communication and computation operations should be scheduled precisely meanwhile [3]. And the scheduling will become more complicated for achieving parallelization freedom.

The fluid scheduling on the clustered many-core architectures in this paper involves (1) how to determine parallelization of tasks and assign cores properly; (2) how to assign proper memory to datum; (3) how to generate a validate schedule among task execution, memory access and network communication to meet the time constraints and minimize the system density. We call the problem as "Time Constraints - Density-Aware" Real-Time Fluid Scheduling (TDRFS) problem.

2. Conventional designs and previous work

This paper focuses on the study of fluid scheduling problem, a branch of real-time tasks schedule on parallel processing platforms [9].

Task model has a significant influence on the fluid scheduling performances. Typical models include multi-threaded model [12] and directed acyclic graph (DAG) [6]. Tendulkar et al.[7] have integrated...
these two models and proposed a split DAG model, which can describe the precedence constraints of fluid tasks and support parallel scheduling simultaneously. But this model assumes that the parallelization is pre-fixed. Besides, in order to achieve accurate scheduling on multi-core architectures, Wang et al. [6] have proposed a new DAG model incorporating memory access operations. But this new model is not accurate enough for clustered many-core architectures because it neglects network communication.

Recent researches on parallel scheduling are dedicated to improve application performance by designing parallel scheduling algorithms [7, 14, 4, 5]. However, most researches contained the assumption that the parallelization is pre-fixed too. Moreover, researches on scheduling with time constraints generally focuses on optimizing other QoS indicators. For instance, Kim et al.[8] have presented an efficient fairness-oriented scheduling algorithm which can minimize the difference in task execution performance and obtain high system throughput. Cheng et al.[10] have proposed a parameterized approximate scheduling algorithm to improve the efficient of system resource utilization, but this method required a high system resource consumption. What caused the TDRFS problem unsolved is these techniques above only considered either parallel scheduling or scheduling with time constraints.

The real-time scheduling problems on clustered many-core architectures has proved to be an NP-hard problem [10]. So, Pathania et al.[11] have proposed an optimal greedy algorithm which obviously reduces the run-time scheduling overhead while maintaining optimality. It does not show a good schedualability yet. Various researches have designed the heuristic algorithms to reduce the complexity of the NP-hard problem [6, 9, 10], however the parallelization-freedom scheduling problem is still unresolved.

This paper puts forward a new task model and an integer linear programming scheduling model combining computation, memory access and interconnection communication. Then a heuristic algorithm about polynomial complexity is presented in order to solve the TDRFS problem. Our technique proves to be able to effectively balance the application execution time and system density. Meanwhile it could improve the schedulability and validity too.

3. The models

3.1. Architecture model

The architecture model is a clustered many-core processor system shown in Fig.1, consisting of a global shared memory and a set of connected clusters C. Each C is composed of local memory and m homogeneous computing cores. Intra-cluster communication are implemented through local memory, while inter-cluster communication are realized through the network controlled by DMA.

The delay of network communication is characterized by $D_{c} = D_{i} + D_{u} \times L_{d}$, $D_{i}$ is fixed initial delay, $D_{u}$ is the delay per byte, and $L_{d}$ is the size of transmitting data in bytes. Both computing cores and DMA execute initialization phase $D_{i}$, only DMA executes the data transmitting phase denoted as $D_{tr} = D_{u} \times L_{d}$. This paper applies asynchronous transmission mechanism among clusters to achieve efficient data transmit, so feedback signals is indispensable.

An architecture model can be abstracted to a tuple as $A = \{C, In, R_{i}, R_{s}, M_{i}, D_{i}, D_{o} \mid C_{i} = \{P_{j} \} \cup M_{i}, 1 \leq i \leq n, 1 \leq i \leq m\}$, in which $In \subseteq C \times C$ is the inter-cluster interconnection network. $R_{i}$ and $R_{s}$ are the capacity of the local memory and DMA channel respectively. $M_{i}$ is the maximum of concurrent memory access operations.

3.2. Task model

In order to improve the validity of our approach, we consider memory access and network communication in succession to build an accurate task model.
Definition 1 (Memory access task graphs): memory access task graph MG is a tuple as $(V_p, V_m, E, D, W, acs, \alpha)$. $V_p$ is the task actor; $V_m$ is the memory access actor; $D$ is the data set which would be executed in tasks; $W(v_k)$ is the delay of task $v_k$; $E_i \subseteq V \times V_i$ ($V_i = V_p \cup V_m$) are the edges representing data dependencies between task actors; $acs : V_p \times V_m \times D \rightarrow \{0, 1\}$ is a binary function, e.g., $acs(v_k, u_l, d_h)$ denotes whether memory access operation $u_l$ transmits datum $d_h$ to task $v_k$. $\alpha$ is the parallelization factor, i.e., the task $v_k$ can be split into $\alpha(v_k)$ threads. Fig. 2(a) gives the input/output data and execution delay of each task, and the traditional DAG is shown in Fig. 2(b). Fig. 2(c) is the MG derived from Fig. 2(b).

Definition 2 (Communication task graphs): a communication task graph CMG is a tuple as $(V_p, V_m, V_r, V_i, E_z, D, W, acs, \alpha)$. $V_i$ and $V_r$ are both network communication actors, representing the initial actor and data transmitting actor. $E_z \subseteq V_z \times V_z$ ($V_z = V_p \cup V_m \cup V_i \cup V_r$) are edges representing the data dependencies between any two actors. Assuming that task $c$ and task $d$ transmit data through the network, the local memory capacity $R_1$ and the DMA channel capacity $R_2$ are both 3. We first introduce the communication scheduling diagram as shown in Fig. 3, in which $V_F$ is the query operation actor required for asynchronous transmission. We then get CMG as shown in Fig. 2(d). $V_r$ and $V_i$ are the read and write child actor from $V_i$, and $V_{rtr}$ and $V_{rw}$ are from $V_r$. 

![Figure 1. The architecture of clustered many-core processor.](image1)

![Figure 2. Distributions of tasks.](image2)
Figure 3. Communication scheduling diagram.

Definition 3 (System density): Assuming that the generation time and relative deadline of a real-time task \( v_k \) are \( t_{\text{start}} \) and \( t_{\text{rend}} \), the absolute deadline \( t_{\text{rend}} \) can be presented as \( t_{\text{rend}} = t_{\text{start}} + t_{\text{rend}} \). It is obvious that \( t_{\text{start}}(v_k) \leq \min\{t_{\text{rend}}(th_{ks})\} \) and \( \max\{t_{\text{rend}}(th_{ks})\} \leq t_{\text{rend}}(v_k) \). Define \( s_i(th_{ks}, t) \) as the state function of thread \( th_{ks} \):

\[
s_i(th_{ks}, t) = \begin{cases} 1, & t_{\text{start}} \leq t \\ 0, & 0 < t < t_{\text{start}} \\ t_{\text{rend}} < t \leq T, \end{cases} \quad k \in [1, p], s \in [1, \alpha(v_k)]. \tag{1}
\]

∀th_{ks} \in v_k, if \( s_i(th_{ks}, t) = 1 \), thread \( th_{ks} \) will be active at time \( t \) and the task \( v_k \) will be active too. The computing density of thread \( th_{ks} \) is defined as \( \text{den}_k(th_{ks}) = t_{\text{rend}}(th_{ks}) / t_{\text{rend}}(v_k) \). And the system density is defined as the sum of the computing density of all active threads:

\[
\text{den}_\text{sys}(t) = \sum_{s=1}^p \sum_{t=1}^{\alpha(v_k)} s_i(th_{ks}, t) \times \text{den}_k(th_{ks}), t \in [1, T]. \tag{2}
\]

4. Integer linear programming model
An integer linear programming (ILP) model is established to generate a schedule and the parallelization factor of all tasks. The constraints of this model include resource, execution time and data dependency constraints; the objective is to minimize system density.

4.1. Computing resource constraints
The task assignment is modeled as two binary variables \( x_{ks, i, j, t} \) and \( y_{ks, i, j, t} \), representing thread \( th_{ks} \) starts or not and whether it is scheduled at time \( t \) respectively. \( \text{CL}: th_{ks} \rightarrow \text{const}, \text{const} \in [1, 2, \cdots, n] \) is a function—denotes which cluster \( th_{ks} \) belongs to. The computing resource constraints can be formulated by Eq. (3). Each thread starts execution on only one core (Eq. (3.a)) and there is at most one thread is executing anytime on each core (Eq. (3.b)). Besides, multiple parallel threads spited from same task \( v_k \) must belong to the same cluster due to the high communication overhead (Eq. (3.c)).

4.2. The storage resource constraints
The storage resource constraints include two parts: local memory constraints and DMA channel constraints. The local memory is divided into \( m \) banks to resolve competition problem corresponding to \( m \) cores. \( \text{bank}_{i,j} \) only can be written by \( P_{i,j} \), but every core in cluster \( C_i \) possess authority to read it. The binary variable \( y_{k, i, j} \) denotes whether datum \( d_{k, i, j} \) is allocated to memory \( M_i \). \( y_{i, j, l} \) and \( y_{i, j, l} \) denote whether memory access operation \( u_l \) starts and whether it is scheduled in memory \( M_l \) at time \( t \) respectively. For each local memory \( M_i \), the size of all data in \( M_i \) must be less than or equal to the
memory capacity \( R_i \) (Eq. (4.a)). Similar to task execution operation, each memory access operation \( u_l \) start executing for only one time (Eq. (4.b)). And whenever the number of memory access operation actors are bounded by the concurrent access number \( M_c \) of local memory (Eq. (4.c)).

\[
\sum_{k=1}^{n} \sum_{i=1}^{m} x_{k,i,j,t}^2 \leq 1, i \in [1,n], j \in [1,m], t \in [1,T] (b) 
\]

\[
C(ks) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} i \times x_{k,i,j,t}^1 = \text{const}, s \in [1,\alpha(v_k)], \text{const} \in Z(c) 
\]

4.3. The data dependencies constraints

In CMG, the function \( W(f) \) represents the delay of actor \( f, f \in V_2 \). The binary variable \( z_{f,j,j,t} \) represents whether actor \( f \) starts. Set \( (f_1, f_2) \) represents the data dependency between \( f_1 \) and \( f_2 \)—\( f_2 \) starts only if \( f_1 \) has finished (Eq. (6)).

\[
\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} (t + W(f_i)) \times z_{f_i,j,j,t} \leq \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{t=1}^{T} i \times z_{f_i,j,j,t} . 
\]

4.4. Time constraints

According to the definition in 3.1, the delay of thread \( th_k \) and task \( v_k \) can be presented as Eq. (7.a) and Eq. (7.b). Assuming that the tasks on the critical path constitute set \( C_{path} \). Eq. (7.c) describes the delay of the application because it is equal to the delay of \( C_{path} \)—the maximum execution time among all possible paths.

\[
T(th_k) = \sum_{i=1}^{T} \sum_{j=1}^{m} \sum_{t=1}^{T} x_{k,i,j,t}^2, k \in [1,p], s \in [1,\alpha(v_k)](a) 
\]

\[
T(v_k) = \max (T(th_k)), s \in [1,\alpha(v_k)](b) . 
\]

\[
T(a) = T(C_{path}) = \sum_{v_k \in C_{path}} T(v_k)(c) 
\]

Assuming that the given time constraint is \( T_d \). According to Eq. (7), the time constraint can be modeled as:

\[
\sum_{v_k \in C} \max \left( \sum_{i=1}^{T} \sum_{j=1}^{m} \sum_{t=1}^{T} x_{k,i,j,t}^2 \right) \leq T_d, s \in [1,\alpha(v_k)]. 
\]
4.5. Objective function
The lower the system density is, the lower the system resource consumption is. The objective of this paper is minimizing system density (Eq. (9)) so as to complete more tasks within the constrained time.

\[
\min \text{den}_{sys}(t) = \sum_{k=1}^{p,\alpha(v_k)} s(t_{th_k}) \times \text{den}_{in}(t_{th_k}), t \in [1, T].
\] (9)

5. Heuristic algorithm
The proposed heuristic algorithm in this section is able to solve the TDRFS problem within polynomial time. \(\Delta \text{sysden}\) represents the increment of system density as the parallelization factors change. And \(\Delta T_a\) represents the increment of application execution time when the data storage locations change. Here, we define two scheduling trade-off factors (Eq. (10)). \(\beta\) is the ratio of \(\Delta \text{sysden}\) to \(\Delta T_a\) when the parallelization factor of task \(v_k\) changes from \(\alpha(v_k)\) to \(\alpha(v_k) '\); \(\gamma\) represents the ratio of \(\Delta \text{sysden}\) to \(\Delta T_a\) when data \(d_a\) moves from \(\text{bank}_i\) to \(\text{bank}_j\).

\[
\beta(\alpha(v_k)) = \frac{\Delta \text{sysden}(\alpha(v_k))}{\Delta T_a(\alpha(v_k))} = \frac{\text{sysden}(\alpha(v_k)) - \text{sysden}(\alpha(v_k) ')}{T_a(\alpha(v_k)) - T_a(\alpha(v_k) ')}. (a)
\]
\[
\gamma(d_a, \text{bank}_j) = \frac{\Delta \text{sysden}(d_a, \text{bank}_j)}{\Delta T_a(d_a, \text{bank}_j)} = \frac{\text{sysden}(d_a, \text{bank}_j) - \text{sysden}(d_a, \text{bank}_j) '}{T_a(d_a, \text{bank}_j) - T_a(d_a, \text{bank}_j) '}. (b)
\] (10)

The details of heuristic algorithm is shown in Algorithm I.

| Algorithm I |
|-------------|
| **Require:** (1)CMG; (2)A; (3)deadline; \(T_d\) |
| 1: initial \(\alpha\) and \(T_{\text{Cpath}}\) to maximum |
| 2: if \(T(\text{Cpath}) > T_a\) then |
| 3: fresh the layout |
| 4: obtain a partition according to [12] |
| 5: map the one with most communication with other partitions to an idle cluster |
| 6: repeat |
| 7: find a partition with highest communication with the partitions mapped |
| 8: find an idle cluster with minimal Manhattan Distance with the mapped clusters |
| 9: all partitions are mapped |
| 10: obtain a list schedule with \(D \rightarrow M\) and \(V_p \rightarrow P\) |
| 11: for each bank, of each cluster do |
| 12: if bank is a new bank and is not full and \(\Delta(d_a, \text{bank}_j) < 0\) then |
| 13: compute \(\gamma(d_a, \text{bank}_j)\) |
| 14: end if |
| 15: end for |
| 16: select \((d_a, \text{bank}_j) \leftarrow \text{minimal}\} \{\gamma(d_a, \text{bank}_j)\} \}
| 17: end if |
| 18: else then |
| 19: repeat |
| 20: if there is \(v_i \notin \text{Cpath}\) and \(\alpha(v_i) > 1\) do |
| 21: \(\alpha(v_i) = \alpha(v_i) - 1\) |
| 22: compute \(\beta(\alpha(v_i))\) |
| 23: end if |
| 24: if there is \(v_i \in \text{Cpath\} and \alpha(v_i) > 1\) do |
| 25: \(\alpha(v_i) = \alpha(v_i) - 1\) |
| 26: compute \(\beta(\alpha(v_i))\) |
| 27: end if |
| 28: end if |
| 29: end for |
| 30: \(v_j \leftarrow \text{minimal}\} \{\beta(\alpha(v_i))\} \}
| 31: end if |
| 32: until all \(\alpha\) cannot be reduced or time is out |

Figure 4. Algorithm I.

We first initial all parallelization factors and the delay of critical path \(T(\text{Cpath})\) are maximum. If \(T(\text{Cpath})\) is smaller than the time constraint \(T_d\), the system density would be reduced by cutting down tasks parallelization factors until they reach their minimum or the calculation is overtime. The selected task should satisfy \(\Delta \text{sysden} < 0\) and have the smallest \(\beta\) for reducing the system density with a smaller increase in execution time. In addition, the tasks on the non-critical path is preferentially changed because the delay of tasks on \(\text{Cpath}\) affects the application delay directly. If \(T(\text{Cpath})\) is bigger than \(T_d\),
$T(C_{path})$ will be reduced by changing the storage location of the data until $T(C_{path})$ is less than $T_d$. Similarly, the selected data should satisfy $\Delta T_a < 0$ and have the smallest $\gamma$. The time complexity of Algorithm I is $O(mV(n\log n + V \log V + 2mE))$. $n$ and $m$ represent the number of clusters and the number of cores in each cluster; $V$ and $E$ represent the number of task actors and edges in DAG.

6. Experiment and evaluation

We implement the experiments for CMGs on the architecture model defined in section 2.1. The parameters of the homogeneous cores refer to Kalray MPPA [18], in which the frequency is 400 MHz and the local memory is 128KB. All the experiments are conducted by the simulator—Intel Core i7 processor at 1.73 GHz with 4 GB of memory.

6.1. Effectiveness evaluation

To show the effectiveness of the proposed algorithm, we use benchmarks shown in Table 1, consisting of DSPstone [13] and a subset of StreamIt benchmarks [14].

Table 1. Application benchmarks.

| Benchmarks | DctCorase | Dct3 | Dct4 | Dct8 | Fft | Lattic-iir |
|------------|-----------|------|------|------|-----|------------|
| Tasks      | 3         | 12   | 21   | 38   | 96  | 260        |
| Edges      | 3         | 11   | 24   | 52   | 112 | 221        |
| Transfer Data(bytes) | 512 | 1024 | 1536 | 2304 | 6144 | 7936 |

We implement all three algorithms (ILP, greedy [15] and Algorithm I) within the same scheduling framework to make reasonable comparisons. The greedy algorithm proves to be effective in scheduling according to the available documents. We adjust it to make it fit for solving the TDRFS problem.

Table 2. The schedule results of the three algorithms.

| Benchmarks | Td/10^4 | ILP density | Greedy density | Algorithm I Density | ILP↑ | Greedy↑ |
|------------|---------|-------------|----------------|---------------------|------|--------|
| JpegDec    | 8       | 110.67      | 120.77         | 113.14              | 2.23%| -6.32% |
|            | 12      | 92.41       | 112.68         | 95.62               | 3.47%| -15.14%|
|            | 16      | 80.06       | 92.56          | 80.83               | 0.96%| -12.67%|
| Dct3       | 13      | 109.02      | 118.36         | 111.21              | 2.01%| -6.04% |
|            | 19      | 90.51       | 102.28         | 92.85               | 2.58%| -9.22% |
|            | 25      | 72.71       | 86.53          | 74.21               | 2.06%| -14.24%|
| Dct4       | 19      | 114.91      | 124.94         | 117.61              | 2.35%| -5.87% |
|            | 27      | 90.89       | 103.22         | 93.45               | 2.82%| -9.47% |
|            | 35      | 65.11       | 77.84          | 65.75               | 0.99%| -15.53%|
| Dct8       | 26      | 109.21      | 130.34         | 113.85              | 4.25%| -12.65%|
|            | 38      | 99.21       | 118.67         | 102.31              | 3.12%| -13.78%|
|            | 50      | 76.80       | 90.62          | 77.59               | 1.03%| -14.38%|
| Fft        | 65      | 106.23      | 122.34         | 109.88              | 3.44%| -10.18%|
|            | 83      | 89.34       | 105.67         | 90.54               | 1.69%| -14.32%|
|            | 101     | 68.98       | 82.62          | 70.33               | 1.96%| -14.87%|
| Lattic-iir | 162     | --          | 132.28         | 112.45              | --   | -14.99%|
|            | 240     | --          | 106.66         | 82.53               | --   | -22.02%|
|            | 318     | --          | 87.95          | 66.25               | --   | -24.67%|
| average    | --      | --          | --             | --                  | 2.33%| -13.16%|

Table 2 shows the schedule results of the three algorithms, in which $T_d$ represents the given time constraints, "--" means that the algorithm cannot produce a legal solution within the given time constraints.
The proposed algorithm reduces the density by 13.16% averagely compared with the greedy algorithm. And the ILP model only performs slightly better than the proposed algorithm 2.33% about density, but its computation time grows exponentially as the size of benchmark increases. For instance, the ILP model takes 149 minute to generate the solution for “Fft” when the given time bound is 65 minutes. But the proposed Algorithm I only requires less than 1.5 minutes to generate a near optimal solution, which demonstrates the effectiveness of Algorithm I. In addition, the advantages of Algorithm I are more pronounced compared with greedy algorithms as time constraints increase.

6.2. Schedulelability evaluation
We compare the proposed approach with the following three approaches to prove the improvement of schedulability. (1) Fully parallelization: The parallelization of each task is up to its maximum. (2) Single-thread: The parallelization factor of each task is 1. (3) This approach comes from Jihye et al.[16]: It only involves the trade-off between execution time and computing density of a single task.

We generate 1500 task sets, and the details of the task sets generation method are as follows. The fluid task set TS consists of \( p \) tasks which is randomly picked from \([2, 20]\). We determine the delay of the task \( v_k \) like the scheme described in reference [9]. And we introduce two factors: the parallelization overhead factor \( \theta \) and the thread difference factor \( \delta \) to get the delay of \( \theta v_k \). The average execution time of threads \( T_{\text{av}}(\theta v_k) \) is assumed as \((1+\alpha (v_k)) \times T(v_k)\). According to this formula, the task has only one single thread and \( T_{\text{av}}(\theta v_k) \) equals to \( T(v_k) \) when \( \theta = 0 \), indicating no parallelization overhead; When \( \theta = 1 \), the parallelization overhead is up to maximum and \( T_{\text{av}}(\theta v_k) \) equals to \( T(v_k) \) again, which indicates the parallelization achieves no performance improvement. The value of \( \theta \) is randomly generated in \([0, 0.3]\), referring to the experiment results in reference [9]. The execution time \( T(\theta v_k) \) of each thread is \( \delta \times T_{\text{av}}(\theta v_k) \), \( \delta \) is randomly generated from \([0.8, 1.2]\) and satisfies the constraint: \( \sum T(\theta v_k) > T(v_k) \). In addition, the time constraint \( T_d \) of TS is determined by the time constraint factor \( \varepsilon \): \( T_d = \varepsilon \times \sum T(v_k) \), \( \varepsilon \in [0.2, 3] \). Fig. 5 shows the schedulability change of the four approaches as \( \varepsilon \) varies in \([0.2, 3]\).

![Figure 5. Schedulable ratio according to the time constraint factor \( \varepsilon \).](image)

When \( \varepsilon = 1 \), the success rate of our scheduling method increased by 1.77, 3.22 and 3.87 times comparing to the other three approaches. The schedulability of ours and the method reported in ref. [16] are higher while \( \varepsilon \) is changing within \([0.2, 3]\). The reason is these two methods can change the parallelization of the tasks according to the task execution delay and computing density. Our method is superior because we take the system density into consideration while ref. [16] only involves the single-task computing density. Besides, when \( \varepsilon \) is small, the schedulability of the Full parallelization is better.
than Single-thread. That is because when $\epsilon$ is small, the time constraint becomes the bottleneck of the schedulability. Full parallelization generates a shorter execution delay. However, when $\epsilon$ is large, the computing density becomes the bottleneck instead, and the result is just the opposite.

To further illustrate the improvement of our approach in trade-off capability, the distribution of task sets is shown in Fig. 6 within the two-dimensional space of execution delay and computing density. The x-axis represents the portion of time bound, and y-axis represents the portion of density bound. The success rate of our scheduling is 73.2% (1098 tasks are schedulable totally), which is 1.68, 3.06 and 12.4 times comparing to the other three technologies respectively. Fig. 6 shows that Single-thread approach results in longer execution delay (94.1% of the tasks are out of the time bound), while Full parallelization brings a larger computing density (63.1% of the tasks are out of the density bound). The approach in ref. [16] leads to an unsatisfactory result in which 43.4% of the tasks exceed both the time and density constraints. This approach is second only to ours in all 4 methods. The experimental results show that our method has a better balance between computing time and density.

![Figure 6. Distributions of tasks.](image)

6.3. Validity evaluation

We compare the validity of our task model (CMG) with DAG and MG by adjusting the ILP model according to DAG and MG respectively. Each application in Table I was executed more than 100 times on the platform in ref. [18]. We measure their real completion time and compare them with the predicted completion time. According to ref. [17], we have obtained different partitioning and layout solutions, from which we can get different scheduling methods further. Fig. 7 shows the total number of the scheduling solutions and the maximum error—the predicted completion time vs the real completion time of different approaches.

The average maximum error of the proposed approach is only 9.69% (as low as 4.23% at best). The average maximum error of DAG and MG models are 27.29% and 21.24%, which are 2.81 and 2.19 times than ours. This is because our approach takes both memory access and network communication operations into account, which are necessary for clustered many-core architecture. But the overall maximum error of our approach reaches up to 18.34%. It is mainly due to the lack of consideration for the communication competition when different clusters occupy the same DMA. Besides, the shared memory access competition also lead error to grow.
7. Conclusion

This paper aims to solve the real-time fluid scheduling problem on the clustered many-core platform, including parallelization-freedom scheduling and co-scheduling problem. We construct an accurate fluid task model that incorporates memory access and interconnection communication and propose a kind of optimal integer linear programming (ILP) model further. For the complexity of ILP is exponentially expanded as the size growth of the task, a heuristic algorithm for polynomial complexity is proposed. The experiment results show that the proposed approach can observably reduce the system computing density and achieve a good trade-off balance between application execution time and system computational density. It could also effectively improve schedulability and validity. For further research, the DMA resource and the shared resource competition problem would be considered in the scheduling to improve the schedulability of our approach.

References

[1] Krishnakumar, A. et al. (2020) Runtime Task Scheduling Using Imitation Learning for Heterogeneous Many-Core Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 39(11): 4064-4077.
[2] Zhenyang, L., Xiangdong L., and Jun, L. (2020) Scheduling Parallel Real-Time Tasks for Multicore Systems. In: 2020 International Conference on Service Science (ICSS). Xining, China. pp. 102-106.
[3] Koike, R., Fukunaga, T., Igarashi, S. and Azumi, T. (2020) Contention-Free Scheduling for Clustered Many-Core Platform. In: 2020 IEEE International Conference on Embedded Software and Systems (ICESS). Shanghai, China. pp. 1-8.
[4] Faure, A., Lucarelli, G., Richard, O. and Trystram, D. (2020) Online Scheduling with Redirection for Parallel Jobs. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). New Orleans, LA, USA. pp. 1-4.
[5] Benoit, A., Le, Févre V., Raghavan, P., Robert, Y. and Sun, H. (2020) Design and Comparison of Resilient Scheduling Heuristics for Parallel Jobs. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). New Orleans, LA, USA. pp. 567-576.
[6] Wang, Y., Li, K., Chen, H., He, L. and Li, K. (2014) Energy-Aware Data Allocation and Task Scheduling on Heterogeneous Multiprocessor Systems With Time Constraints. IEEE Transactions on Emerging Topics in Computing, 2: 134-148.
[7] Tendulkar, P., Poplavko, P., Galanommatitis, I. and Maler, O. (2014) Many-Core Scheduling of Data Parallel Applications Using SMT Solvers. In: 2014 17th Euromicro Conference on Digital System Design. Verona, Italy. pp. 615-622.
[8] Kim, C. and Huh, J. (2018) Exploring the Design Space of Fair Scheduling Supports for Asymmetric Multicore Systems. IEEE Transactions on Computers, 67: 1136-1152.
[9] Kim, K., Cho, Y., Eo, J., Lee, C. and Han, J. (2018) System-Wide Time versus Density Tradeoff in Real-Time Multicore Fluid Scheduling. IEEE Transactions on Computers, 67: 1007-1022.

[10] Cheng, S., Chang, C., Chen, J., Kuo, T. and Hsiu, P. (2016) Many-Core Real-Time Task Scheduling with Scratchpad Memory. IEEE Transactions on Parallel and Distributed Systems, 27: 2953-2966.

[11] Pathania, A., Venkatramani, V., Shafique, M., Mitra, T. and Henkel, J. (2017) Optimal Greedy Algorithm for Many-Core Scheduling. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36: 1054-1058.

[12] Gai, K., Qiu, M., Zhao, H. and Qiu, L. (2016) Smart Energy-Aware Data Allocation for Heterogeneous Memory. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). Sydney, NSW, Australia. pp. 136-143.

[13] Zivojnovic, V. (1994) Dspstone: A DSP-oriented Benchmarking Methodology. In: Proc. Signal Processing Applications & Technology. Dallas, TX.

[14] Frederick, T. W., & Amarasinghe, S. (2009). Language and compiler support for stream programs. Massachusetts Institute of Technology.

[15] Kang, Q., Hong, H., and Song, H. (2011) Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm. Journal of Systems and Software, 84(6): 985-992.

[16] Kwon, J., Kim, K. W., Paik, S., Lee, J., and Lee, C. G. (2015) Multicore scheduling of parallel real-time tasks with multiple parallelization options. In: 21st IEEE Real-Time and Embedded Technology and Applications Symposium. Seattle, WA, USA. pp. 232-244.

[17] Cotton, S., Maler, O., Legriel J., and Saidi S. (2011) Multi-criteria optimization for mapping programs to multi-processors. In: 2011 6th IEEE International Symposium on Industrial and Embedded Systems. Vasteras, Sweden. pp. 9-17.

[18] Dinechin, B. D., Ayrrignac, R., Beaucamps, P. E., Couvert, P. and Massas, P. D. (2013) A clustered manycore processor architecture for embedded and accelerated applications. In: 2013 IEEE High Performance Extreme Computing Conference (HPEC). Waltham, MA, USA. pp. 1-6.