International Journal of Geo-Information

MDPI

Article

Hypergraph+: An Improved Hypergraph-Based Task-Scheduling Algorithm for Massive Spatial Data Processing on Master-Slave Platforms

Bo Cheng [1,2], Xuefeng Guan [1,2,*], Huayi Wu [1,2] and Rui Li [1,2]

[1] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, 129 Luoyu Road, Wuhan 430079, China; chengbo@whu.edu.cn (B.C.); wuhuayi@whu.edu.cn (H.W.); ruili@whu.edu.cn (R.L.)

[2] Collaborative Innovation Center of Geospatial Technology, 129 Luoyu Road, Wuhan 430079, China

* Correspondence: guanxuefeng@whu.edu.cn; Tel.: +86-27-6877-8311

Academic Editor: Wolfgang Kainz

Received: 20 May 2016; Accepted: 29 July 2016; Published: 10 August 2016

Abstract: Spatial data processing often requires massive datasets, and the task/data scheduling efficiency of these applications has an impact on the overall processing performance. Among the existing scheduling strategies, hypergraph-based algorithms capture the data sharing pattern in a global way and significantly reduce total communication volume. Due to heterogeneous processing platforms, however, single hypergraph partitioning for later scheduling may be not optimal. Moreover, these scheduling algorithms neglect the overlap between task execution and data transfer that could further decrease execution time. In order to address these problems, an extended hypergraph-based task-scheduling algorithm, named *Hypergraph+*, is proposed for massive spatial data processing. *Hypergraph+* improves upon current hypergraph scheduling algorithms in two ways: (1) It takes platform heterogeneity into consideration offering a metric function to evaluate the partitioning quality in order to derive the best task/file schedule; and (2) It can maximize the overlap between communication and computation. The GridSim toolkit was used to evaluate *Hypergraph+* in an IDW spatial interpolation application on heterogeneous master-slave platforms. Experiments illustrate that the proposed *Hypergraph+* algorithm achieves on average a 43% smaller makespan than the original hypergraph scheduling algorithm but still preserves high scheduling efficiency.

Keywords: task scheduling; *Hypergraph+*; spatial data processing; master-slave platforms

1. Introduction

In recent years, with the rapid development of surveying and remote sensing technologies, the volume of spatial data has increased dramatically [1–3]. Spatial data processing is a typical type of data-intensive applications where users must access and process massive spatial data. Figure 1 depicts a typical data-intensive computing scenario comprised of a set of storage and computing nodes that collaborate in a network. Each task requires a subset of input files from the storage nodes; a task may share a number of files with other tasks, while an individual task is submitted to one computing node for execution. The computing nodes themselves are connected to the storage nodes for data transfer through a network. This collaboration is orchestrated by a task/data scheduling strategy; therefore, scheduling strategy efficiency has an important influence on collaboration performance.

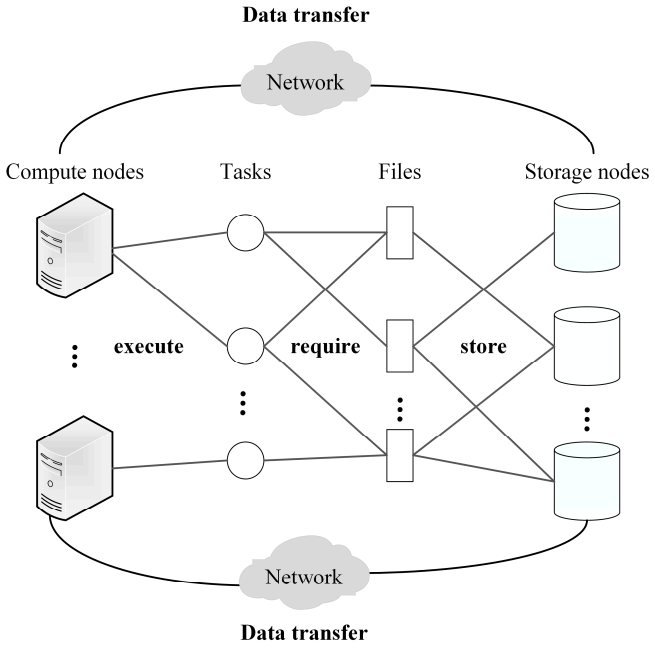


Figure 1. A typical data-intensive computing scenario.

For such data-intensive applications, a number of scheduling strategies have been proposed, including task-oriented, data-aware, and hypergraph-based algorithms. Among these scheduling algorithms, only the hypergraph-based type of algorithms can fully capture data sharing among tasks and thus minimize the overall data transfer while still maintaining a balanced distribution of computing loads across the nodes. Heterogeneous processing platforms, however, generate additional problems in these hypergraph-based scheduling strategies. The formulated hypergraph model can completely represent the relationship among tasks, data files and compute platforms, but as the task execution node and file transfer destination are unknown before scheduling, these types of improved hypergraph algorithms cannot take processors or network heterogeneity into consideration. Without due consideration of platform heterogeneity, scheduling with single hypergraph partitioning is not optimal. Furthermore, the existing scheduling algorithms including hypergraph approaches generally neglect the overlap between task executions and data transfers. These overlooked overlaps might be exploited to further decrease total task execution time.

To address these problems, we propose an extended hypergraph-based task-scheduling algorithm, named *Hypergraph+*. *Hypergraph+* firstly encapsulates a master-slave platform, spatial data processing applications, and a scheduling objective into a general hypergraph model. The later *Hypergraph+* scheduling contains two consecutive stages: matching and ordering. In the matching stage, a *Fitness* function represents platform heterogeneity, evaluates the quality of hypergraph partitioning, and selects the optimum partition. In the ordering stage, a *Sharing-Files* metric determines the task execution in order to maximize overlap between communication and computation.

We conducted experiments to compare our proposed *Hypergraph+* algorithm with three classical scheduling algorithms on a virtual heterogeneous master-slave platform using the GridSim simulation toolkit [4]. These classical scheduling algorithms include *MinMin* [5], *XSufferage* [6], and the pure hypergraph-based scheduling algorithm [7] that we term *Hypergraph* in this paper for the sake of simplicity. The target application is a real IDW interpolation of a massive point cloud. Simulation results illustrate that in comparison to *Hypergraph*, our proposed *Hypergraph+* algorithm can decrease task execution time by more than 43% when scheduling massive spatial data processing applications.

The rest of this paper is organized as follows. Hypergraph partitioning and scheduling strategies for data-intensive applications are introduced in Section 2. The formulated general hypergraph model

for task scheduling is presented in Section 3. Section 4 describes the proposed *Hypergraph+* algorithm, followed by simulation result details in Section 5. Section 6 concludes the paper.

2. Background and Related Work

2.1. Hypergraph and Hypergraph Partitioning

A hypergraph $H = (V, N)$ is defined as a set of vertices V and a set of hyperedges N that connect those vertices [8]. Each hyperedge $n_j \in \mathrm{N}$ is a non-empty subset of vertices V, i.e., $n_j \subseteq \mathrm{V}$. Figure 2 illustrates one hypergraph; in this figure, the closed curve represents one hyperedge and the dots in the closed curve denote the vertices on this hyperedge. A graph can be treated as a special type of hypergraph where each hyperedge can only connect two vertices. Similar to a graph, the weights w_i and costs c_j can be assigned to the vertices ($v_i \in \mathrm{V}$) and hyperedges ($n_j \in \mathrm{N}$) of the hypergraph, respectively.

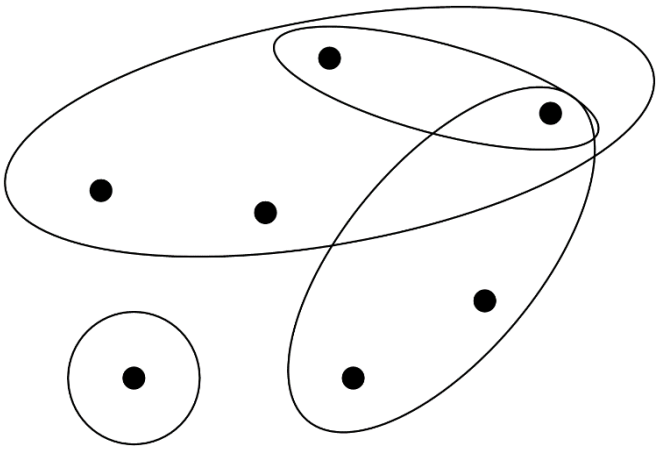


Figure 2. An illustration of one hypergraph: the dots represent the vertices, and the closed curves denote the hyperedges.

A partition $\Pi = \{V_1, V_2, \ldots, V_K\}$ is called a *K-way* partition of H if (1) each part V_k is a non-empty subset of H; (2) all parts are disjointed pairwise and (3) the union of all parts is equal to V. In one partition Π, if a hyperedge has at least one vertex in a part, then it is connected to this part. The connectivity set Λ_j of a hyperedge n_j denotes all the parts connected by n_j, and the connectivity value $\lambda_j = |\Lambda_j|$ of n_j is defined as the number of parts connected by n_j. If a hyperedge connects more than one part, it is cut (i.e., $\lambda_j > 1$), and if otherwise, it is considered as uncut (i.e., $\lambda_j = 1$). The *cutsize* of a partition Π is computed as in Equation (1):

$$\text{cutsize}\,(\Pi) = \sum_{n_j \in N_{cut}} c_j\left(\lambda_j - 1\right) \tag{1}$$

where N_{cut} is the set of all cut hyperedges and each cut hyperedge n_j incurs a cost of $c_j\left(\lambda_j - 1\right)$. This partition *cutsize* is also known as the *connectivity-1* metric.

To solve the hypergraph partition problem, a partition must be found where the *cutsize* is minimized, and a relative balance among all the parts is maintained. A partition Π of H is balanced if the workload W_k of each part V_k satisfies the balance criterion, shown in Equation (2):

$$W_k = W_{avg}\,(1 + \varepsilon), \text{ for } 1 \leqslant \mathrm{k} \leqslant \mathrm{K} \tag{2}$$

where $W_k = \sum_{v_i \in V_k} w_i$ denotes the sum of the vertex weights of one part V_k; $W_{avg} = \frac{\sum_{V_i \in V} w_i}{K}$ is the average weight; and ε is a predetermined imbalanced value.

Although hypergraph partitioning is a NP hard problem, there are still some excellent hypergraph partition algorithms. In addition, open source tools, such as hMETIS [9], PaToH [10], and Parkway [11], are available to implement high-quality hypergraph partitioning. Hypergraph partitioning is widely used in many fields, including very-large-scale integration (VLSI) design [12,13], data mining [14,15], parallel scientific computing [16,17], and task scheduling for data intensive applications [7,18].

Moreover, a hypergraph can be also represented as a bipartite graph, e.g., conceptual graph [19,20]. A conceptual graph contains two disjoint vertices and the semantic relationships as directed edges connecting the disjoint vertices. Conceptual graphs can support problem solving and decision making processes, including artificial intelligence, data mining and case-based reasoning [21]. However, a hypergraph is much more general and obvious than a conceptual graph, and was selected here to model massive spatial data processing.

2.2. The Scheduling Heuristics for Data Intensive Applications

Scheduling data-intensive applications has been extensively studied, and a number of scheduling algorithms have been proposed. According to whether and how they take data transfer into account, these algorithms can be classified into three categories: task-oriented, data-aware, and hypergraph-based.

Task-oriented scheduling algorithms usually require detailed information about tasks and machines for accurate estimation of task execution times on each machine. Maheswaran et al. proposed several typical mapping heuristics including *MinMin*, *MaxMin* and *Sufferage* [5]. From the unscheduled tasks, the *MinMin* chooses the task that has the minimum earliest completion time and allocates this task to a corresponding machine that can compute it the quickest. Unlike *MinMin*, *MaxMin* assigns the task with the maximum earliest completion time to the fastest executing node. *Sufferage* selects the task with the highest *sufferage* value, defined as the difference between its earliest completion time and its second earliest completion time. None of these heuristics, however, considers data issues when making scheduling decisions in data intensive applications and therefore, they are inefficient.

Different from task-oriented algorithms, data-aware scheduling algorithms can produce significant performance improvements as they take both data transfer and task scheduling into account [2,22,23]. Casanova et al. proposed an extension of *Sufferage* called *XSufferage*, which exploits file locality and computes a cluster-level *sufferage* value to achieve better performance [6]. The *Close-to-Files* algorithm [24] schedules tasks with file replication on the least loaded processor close to the sites where the input files are stored. Zhang et al. proposed metaheuristic data pre-scheduling and dynamic task scheduling strategies to solve all-to-all comparison problems in heterogeneous distributed systems [25]. Szmajduch and Kołodziej presented a new version of the Expected Time to Compute Matrix model (ETC Matrix), in which the data transmission and task computation are involved [26]. These data-aware scheduling algorithms however, do not consider file sharing patterns in a global way and thus cannot fully exploit high degrees of shared I/O.

Hypergraph-based scheduling algorithms globally optimize the data transfer during task scheduling. Khanna et al. proposed a hypergraph partitioning-based strategy to schedule a batch of independent tasks to minimize the volume of remote data transfer and contention on storage nodes while maintaining a balanced computational load distribution across compute nodes [7]. Kaya and Aykanat proposed an iterative scheduling approach that improves the scheduling performance by adopting hypergraph-partitioning [18]. They exploit data sharing in a global way to achieve more enhanced performance than the other two types of algorithms.

However, since the file transfer node and task execution destination are unknown, the formulated hypergraph scheduling model cannot fully represent the underlying heterogeneous platforms in which the processors have different processing capabilities and network links have different bandwidths. Hence, a single hypergraph partitioning may not be optimal since platform heterogeneity is neglected. Furthermore, these scheduling algorithms cannot adequately exploit the overlap between communication and computation. Therefore, a new task-scheduling algorithm that can address

the heterogeneous platform problem and maximize the communication-computation overlap is urgently needed.

3. Hypergraph-Based Task Scheduling Model

In this section, we formulate a general hypergraph-based task scheduling model consisting of a master-slave platform, spatial data processing applications, and the scheduling objective.

3.1. Platform Model

The target platform conforms to a typical heterogeneous master-slave paradigm and contains a master P_0 and a set of p slave processors, $P = \{P_1, P_2, \ldots, P_p\}$ as depicted in Figure 3. The master P_0 is connected to slaves over a local area network. The slaves are employed as computing nodes and each has a relative processing capability ρ to execute the tasks. We assume that all the data files are initially stored on the master P_0, so if an input file required by a task is not in the slave processor where the task is executed, it must be requested from the master P_0.

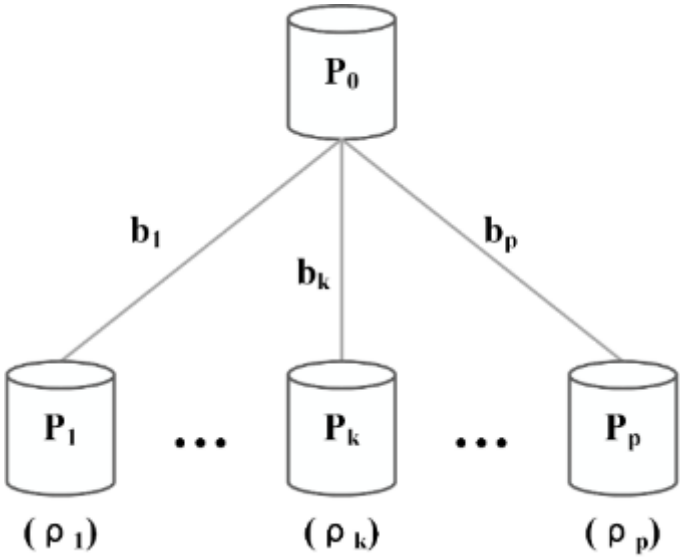


Figure 3. The target heterogeneous platform.

The bandwidth of the link between the master P_0 and the slave processor P_k is denoted by b_k ($k = 1, 2, \ldots, p$), while the maximum outgoing bandwidth of P_0 is denoted by b_m. In order to decrease the waiting time for tasks, task executions and file transfers can overlap on the slaves, i.e., a slave processor can execute a task while accepting the necessary files to execute the next task.

The multiplexed connection model [27] that enables communications between the masters and slaves is used: (1) it allows multiple slaves to download files from the master P_0 simultaneously; (2) two slaves cannot request the same file at the same time; and (3) a slave processor can receive another file after it has saved the previously received file on its local disk.

3.2. Application Model

The spatial data processing application $A = (T, F)$ consists of a set of independent tasks $T = \{t_1, t_2, \ldots, t_n\}$ and m files $F = \{f_1, f_2, \ldots, f_m\}$. The execution of each task t_i depends upon a subset of files, denoted by $F_i = \{f_1, f_2, \ldots, f_k\}$; a given file may be shared by several tasks. The target application A can be represented as a hypergraph model $H = (V, N)$ to capture this data-sharing pattern. In our proposed formulated hypergraph model H, tasks correspond to vertices and files correspond to hyperedges. A hyperedge n_j connecting some vertices means that this file f_j is needed as input and is

shared by a set of tasks. The vertex weight w_i is the estimated completion time of the corresponding task $T_{ct}(t_i)$, and the hyperedge weight c_j is equal to the file size $Size(f_j)$.

The estimated completion time of one task $T_{ct}(t_i)$ is the sum of the total input file transfer time from the master P_0 and the actual task computation time. Prior to task mapping, the file transmission destination is unknown, but the actual file transfer time can be estimated from the size of file f_j divided by the maximum outgoing bandwidth b_m of P_0. For spatial data processing applications, it is feasible to assume the actual computation time of a task is proportional to the size of its input files $F_{i,}$ and C is the predefined computation cost of one data byte. Thus, the total estimated completion time of task t_i will be defined as in Equation (3):

$$T_{ct}(t_i) = \sum_{f_j \in F_i} \left(Size\left(f_j\right)\right) * \frac{1}{b_m} + \sum_{f_j \in F_i} \left(Size\left(f_j\right)\right) * C = \sum_{f_j \in F_i} \left(Size\left(f_j\right)\right) * \left(\frac{1}{b_m} + C\right) \quad (3)$$

Neighborhood computations are usually required in spatial data processing applications. Figure 4 shows some typical neighborhood configurations, including von Neumann, Moore, and extended Moore neighborhoods [28]. In neighborhood processing, one cell generally corresponds to a block of pixels, whose attribute values are stored in a file. When a neighborhood algorithm is used, for example, to calculate slopes and aspects from elevations, the computation task for a given cell requires the values of its neighborhood cells (including the cell itself), i.e., a set of corresponding files.

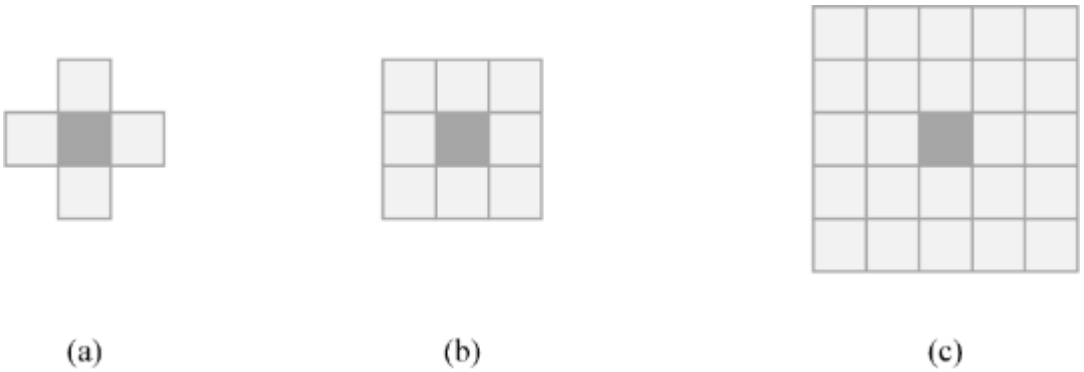


Figure 4. Typical and regular neighborhood configurations: (**a**) Von Neumann neighborhood, (**b**) Moore neighborhood, and (**c**) Extended Moore neighborhood.

A Moore neighborhood example is also presented here to illustrate how to build such a hypergraph model: in Figure 5, tasks T = {1, 2, 3, 4, 5, 6, 7, 8, 9}, and files F = {A, B, C, D, E, F, G, H, I}. The file B, file D, and file E are required by task T_1 to calculate the first cell, then F_1 = {A, B, D, E}. Similarly, F_2 = {A, B, C, D, E, F}, F_3 = {B, C, E, F}, F_4 = {A, B, D, E, G, H}, F_5 = { A, B, C, D, E, F, G, H, I}, F_6 = {B,C, E, F, H, I}, F_7 = {D, E, G, H}, F_8 = { D, E, F, G, H, I}, and F_9 = {E, F, H, I}.

A **T_1**	B **T_2**	C **T_3**
D **T_4**	E **T_5**	F **T_6**
G **T_7**	H **T_8**	I **T_9**

Figure 5. Tasks and files in a Moore neighborhood algorithm.

Figure 6 illustrates the formulated hypergraph model: Hypergraph $H = (V, N)$, $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $N = \{A = \{1, 2, 4, 5\}, B = \{1, 2, 3, 4, 5, 6\}, C = \{2, 3, 5, 6\}, D = \{1, 2, 4, 5, 7, 8\}, E = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}, F = \{2, 3, 5, 6, 8, 9\}, G = \{4, 5, 7, 8\}, H = \{4, 5, 6, 7, 8, 9\}$, and $I = \{5, 6, 8, 9\}$; the master P_0 initially holds all the files, and the slaves $P = \{P_1, P_2, P_3\}$ will execute these tasks.

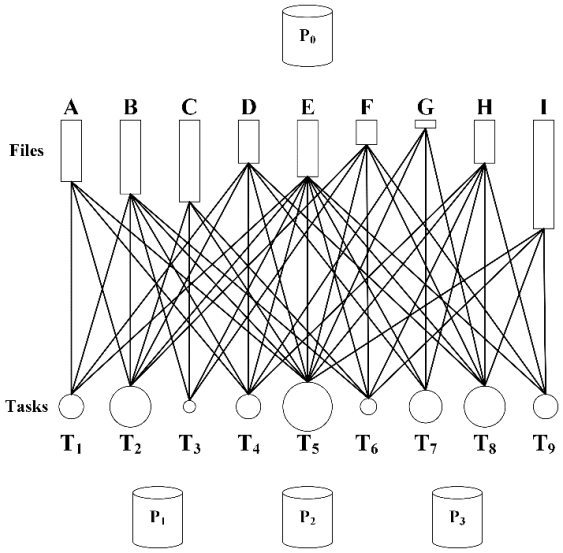


Figure 6. Hypergraph-based task-scheduling model.

3.3. Scheduling Objective

The scheduling objective is to minimize the overall execution time, known as the makespan, which starts from the first file transfer and ends with the completion of the last task execution [18]. Since the estimated completion time of one task is the sum of the total data transfer time and actual task computation time, the scheduling objective is to shorten the amount of data transfer and balance the computational load across the slaves in such a way that the overall execution time is minimized. With the help of the formulated hypergraph model, this objective will be further generalized and considered as the objective of hypergraph partitioning.

Data transfer minimization is achieved by the hypergraph partitioning objective.

After constructing the hypergraph H, the objective of a typical hypergraph partitioning problem is to find a partition $\Pi = \{V_1, V_2, \ldots, V_K\}$ where the *cutsize* is minimized. For a given partition Π, a cut hyperedge n_j with connectivity λ_j means that the file f_j needs to be transferred $\lambda_j - 1$ more times but incurs additional $(\lambda_j - 1) * \text{Size}(f_j)$ bytes of data transmission. Thus, the total communication cost can be computed as in Equation (4):

$$\text{comm}(\Pi) = \sum_{f_j \in F} \lambda_j * Size(f_j) = \sum_{n_j \in N} \lambda_j * c_j = \sum_{n_j \in N_{cut}} (\lambda_j - 1) * c_j + \sum_{n_j \in N} c_j \quad (4)$$

where $\sum_{n_j \in N} c_j$ is equal to total input file size and can be treated as constant, so the comm (Π) depends on the cutsize (Π). Thus, minimizing the *cutsize* is equivalent to minimizing the total data transfer.

The scheduling load-balance is guaranteed by the hypergraph partitioning constraint.

Equation (2) shows that a partition Π of H is balanced if each part V_k satisfies the balance constraint. Since the estimated completion time of a task is the weight of the corresponding vertex, then the workload of one slave processor P_k is equal to the accumulated execution time of all assigned tasks:

$$W_k(\Pi) = \sum_{V_i \in V_k} w_i = \sum_{t_i \in P_k} T_{ct}(t_i) = \sum_{t_i \in P_k} \sum_{f_j \in F_i} \left(\frac{1}{b_m} + C \right) * Size(f_j) \quad (5)$$

Thus, achieving balance among all the grouped vertices during hypergraph partitioning corresponds to balancing the workload of slave processors during the scheduling.

4. The *Hypergraph+* Scheduling Algorithm

The proposed *Hypergraph+* scheduling algorithm has two consecutive stages: matching and ordering. Section 4.1 introduces hypergraph partitioning for mapping tasks to the slave processors. Section 4.2 explains an ordering algorithm that efficiently orders tasks for execution and accordingly transfers the needed files.

4.1. Hypergraph Partitioning for Matching Tasks

Hypergraph partitioning provides an initial scheme to assign tasks to slave processors so that data transfers are minimized and computational workloads are balanced. Single hypergraph partitioning may not be optimal, however, under conditions of platform heterogeneity. Therefore, we consider both network heterogeneity and processor heterogeneity when evaluating the quality of partitioning results for optimization. The whole flow of matching tasks to slaves is shown in Figure 7. First, the input hypergraph model is quickly partitioned with the PaToH tool [10] to obtain partition $\Pi = \{V_1, V_2, \ldots, V_K\}$. Next, the *fitness* evaluation is carried out on partition Π with the *fitness* function: *Fitness*(Π). Finally, the optimum partition is chosen to map tasks to slaves.

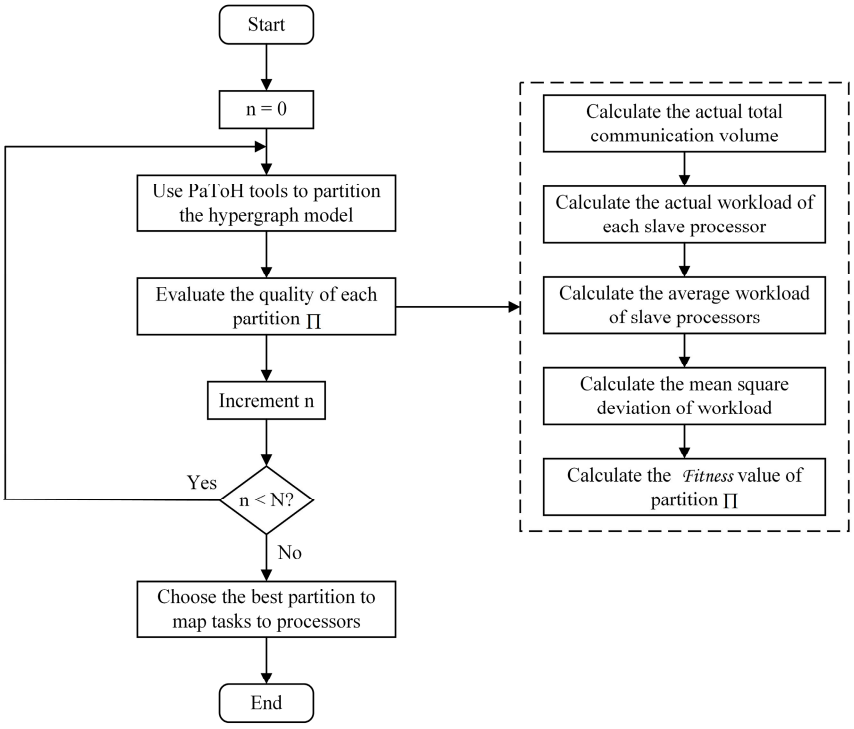


Figure 7. The flow diagram of hypergraph partitioning for matching tasks.

The *Fitness*(Π) evaluation is as follows:

(a) Equation (4) is only valid for homogeneous network cases, and λ_j is set to constant $\frac{1}{b_m}$. After obtaining one hypergraph partitioning, the actual communication volume $\text{comm}'(\Pi)$ is

calculated with Equation (6). The heterogeneous network, λ_j is modified to $\sum_{P_k \in \Lambda_j} \frac{1}{b_k}$, where Λ_j denotes the set of slave processors needed to transfer file f_j, and b_k is the bandwidth between P_k and P_0.

$$\mathrm{comm}'(\Pi) = \sum_{f_j \in F} \sum_{P_k \in \Lambda_j} \frac{Size(f_j)}{b_k} \tag{6}$$

(b) Then, the actual workload of each slave processor $W'_k(\Pi)$ is calculated, which is the sum of the computation load and communication cost. In contrast to Equation (5), relative processing capability ρ_k is added to represent the actual computational load and b_k is substituted for b_m to derive the actual communication load in Equation (7).

$$W'_k(\Pi) = \sum_{t_i \in P_k} \sum_{f_j \in F_i} \left(\frac{1}{b_k} + \frac{C}{\rho_k} \right) * Size(f_j) \tag{7}$$

(c) The average of workload $W'_{avg}(\Pi)$ and the mean square deviation of workload $sd'_{W'_k}(\Pi)$ are calculated as in Equations (8) and (9).

$$W'_{avg}(\Pi) = \frac{\sum_{k=1}^{K} W'_k(\Pi)}{K} \tag{8}$$

$$sd'_{W'_k}(\Pi) = \sqrt{\frac{\sum_{k=1}^{K} \left(W'_k(\Pi) - W'_{avg}(\Pi) \right)}{K}} \tag{9}$$

(d) The *fitness* value is defined in (10): a smaller *fitness* value implies that the partition has lower communication overhead and a more equally balanced computational load.

$$\mathrm{Fitness}(\Pi) = \mathrm{comm}'(\Pi) * sd'_{W'_k}(\Pi) \tag{10}$$

From Equation (10), a lower *fitness* value means a better partition quality. Generally, the lowest *fitness* value is inversely proportional to the repetition number n, but a greater repetition number will increase the entire evaluation time cost. To achieve a cost/quality balance, the iteration number n is chosen as follows. Initially, n is set to a given number (e.g., 10). Then, n doubles each time until the percentage decrease in the lowest *fitness* value is smaller than a given threshold (e.g., 5%). In this way, the evaluation will generate satisfactory partition quality without costing much time.

4.2. Ordering Tasks and File Transfers

After all the tasks have been assigned to their destination processors, *Hypergraph+* will then determine the task execution order and input file transfer so as to maximize the overlap between computation and communication while decreasing the end-point contention among the slaves.

In order to achieve overlap maximization, a *Sharing-Files*(*SF*) metric is introduced to order the task execution on each processor. This metric computes how similar one task is to other tasks. The *SF* value of one task is defined as the number of bytes that the task input files shares with other tasks on the assigned processor. It can be calculated as in Equation (11):

$$\mathrm{SF}(t_i) = \sum_{t_j \in P_k} \sum_{f_x \in F_i \bigcap F_j} Size(f_x), \mathrm{i} \neq \mathrm{j} \tag{11}$$

Task t_i with higher $SF(t_i)$ value means that its input files are shared with more tasks. Task t_i that has the highest $SF(t_i)$ value will be executed first; then, the required files are transferred in advance; thus other tasks relying on these files are subsequently executed. In this case, communication and computation can be overlapped to decrease the waiting time of tasks. Algorithm 1 outlines the proposed task ordering heuristic.

Algorithm 1 Ordering tasks for execution

(1) **for** each slave processor P_k in P
(2) **for** each task t_i mapped to P_k
(3) evaluate Function $SF(t_i)$;
(4) build the list $L(P_k)$ of the tasks sorted in decreasing order of $SF(t_i)$;
(5) **endfor**
(6) **endfor**
(7) **do** until all tasks in T are scheduled
(8) find the slave processor P_k with maximum workload;
(9) **if** (input files of task t_i are already on the processor P_k)
(10) select task t_i to execute;
(11) **else**
(12) select the first task t_i in $L(P_k)$ to execute;
(13) schedule the file transfers of task t_i; (**Algorithm 2**)
(14) remove task t_i from $L(P_k)$, T;
(15) update the workload of P_k;
(16) **enddo**

The transfer of file $f_j \in F_i$ is scheduled based on its earliest completion time. Two tasks on different slave processors may depend upon the same input file and cause end-point contention among slave processors. In order to reduce end-point contention, the estimated completion time includes the actual transfer time and the waiting time. The actual transfer time is the size of file f_j divided by the bandwidth. $T_w(f_j)$ denotes the time spent waiting for transfer in the queue since other slave processors have previously sent requests for the file f_j to the master P_0. The estimated completion time for transferring file f_j is computed as in Equation (12):

$$T_{ct}\left(f_j\right) = \frac{Size\left(f_j\right)}{b_k} + T_w\left(f_j\right) \tag{12}$$

Algorithm 2 describes the general structure of the file transmission heuristic in each processor. The heuristic starts by computing the estimated completion time of each file in F_i, which is the set of files requested by task t_i (line 2 to line 4). It schedules the transfer of file f_j with the earliest completion time (line 5). Then, file f_j is removed from F_i (line 6) and $T_w(\mathrm{f_j})$ is updated accordingly (line 7). This heuristic performs the next iteration of the **do** loop until all files are transferred.

Algorithm 2 The file transmission heuristic in each processor

(1) **do** until all files in F_i are transferred
(2) **for** each file f_j in F_i
(3) compute $T_{ct}(f_j)$;
(4) **endfor**
(5) transfer file f_j with the earliest completion time from P_0 to P_k;
(6) remove file f_j from F_i;
(7) update $T_w(f_j)$;
(8) **enddo**

5. Experiments and Discussion

5.1. Simulated Resources

Simulations provide a repeatable and controllable evaluation environment, and were used to perform an evaluation of our proposed *Hypergraph+* algorithm. We selected the GridSim toolkit [4] to conduct the simulations since it allows us to model heterogeneous processor resources and network connectivity with different bandwidths. GridSim also supports both static and dynamic scheduling simulations.

In this simulation, six slave processors were defined as in Table 1 to execute the input tasks. Each slave processor contains two distinct characteristics, the CPU speed and network bandwidth. Since the task execution time can be defined in terms of million instructions (MI), the CPU resource speed was modeled as million instructions per second (MIPS). The network bandwidth is the bandwidth of the link between the master and the slave. The MIPS and bandwidth were randomly generated for this evaluation experiment [29].

Table 1. Slave setup for the simulation.

Slave	MIPS	Bandwidth
P_1	200	170
P_2	260	320
P_3	160	280
P_4	540	630
P_5	390	470
P_6	410	390

5.2. Experimental Application and Datasets

We selected spatial interpolation as the target application to evaluate the Hypergraph+ scheduling algorithm. For simplicity, inverse distance weighted (IDW) interpolation was used in the experiments. IDW reflects the principle that the estimated value of a cell is more likely correlated with nearby points than distant points [30]. The IDW interpolation equation is defined as,

$$Z_p = \frac{\sum_{i=1}^{k} \left(\frac{Z_i}{d_i^{\beta}} \right)}{\sum_{i=1}^{k} \left(\frac{1}{d_i^{\beta}} \right)} \tag{13}$$

where Z_p is the interpolated value at the target point p; Z_i is the observed value at the *ith* scatter point p_i in the neighborhood of p; k is the number of scatter points taken into the interpolation in the predefined neighborhood of p; d_i is the Euclidian distance from the ith scatter point p_i to p; and β is an arbitrary positive number called the weighting exponent.

A LiDAR point cloud dataset was used as real input in the experiments. These LiDAR point cloud data were acquired in Gilmer County, West Virginia, USA and were free for downloading on the Internet (http://www.wvview.org/data/lidar/Gilmer/). The dataset contains 0.883 billion points, and the point spacing is about 1.4 m, illustrated in Figure 8. This dataset is stored in the ASPRS LAS file format. The total data size is approximately 16.4 GB.

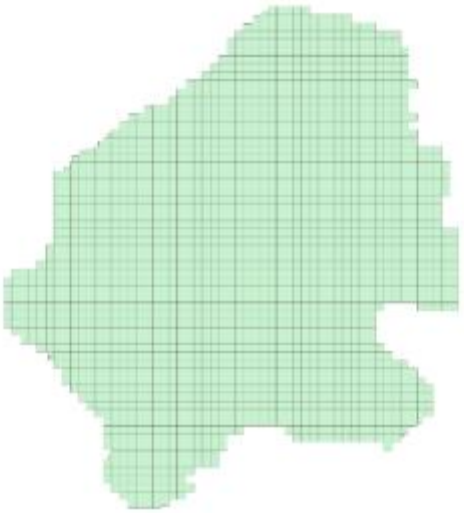


Figure 8. The Gilmer County LiDAR dataset.

The experimental LiDAR dataset was later divided into multiple point blocks. IDW interpolation requires Moore neighborhood to be used as the neighboring blocks input, and the formulated Hypergraph application model is the same as the example model defined in Section 3.2. In this application model, the hyperedge weight c_j was set to each point block size. The vertex weight w_i was set to the interpolation time of the corresponding point block T_{ct}.

In Equation (3), we derived that the actual computation time of one task is proportional to the size of its input files. Thus, an additional experiment was conducted firstly to explore the quantitative relationship between the IDW interpolation time and the input points size. As shown in Figure 9, the relationship between the data size of points and IDW interpolation runtime is almost linear ($R^2 > 0.99$). From the curve fit function, C was solved to 0.0002 for Equation (3).

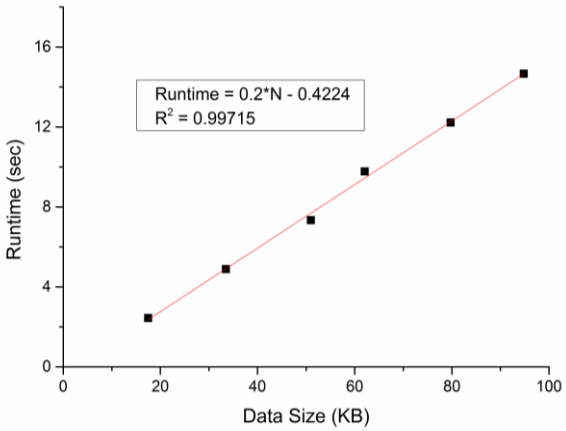


Figure 9. The quantitative relationship between input points size and IDW interpolation time.

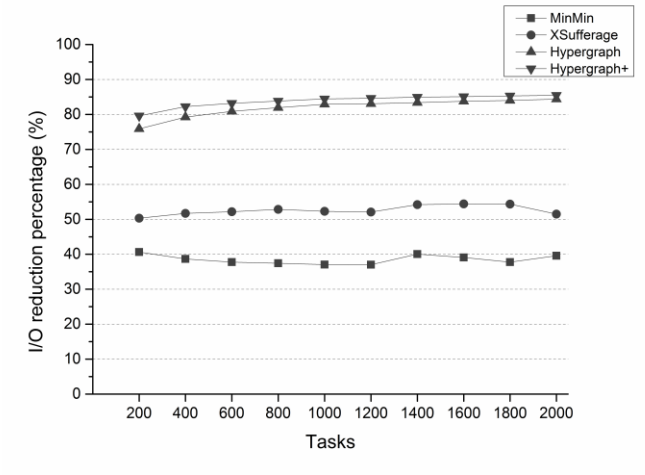
5.3. Evaluation Results and Discussions

With the formulated platform and application models in Sections 5.1 and 5.2, experiments were carried out to evaluate the performance and efficiency of *Hypergraph+*, comparing it to *MinMin* [5], *XSufferage* [6] and *Hypergraph*, which is the original hypergraph partitioning-based approach [7]. These three heuristics are typical task-oriented, data-aware, and hypergraph-based scheduling algorithms described as in Section 2.2.

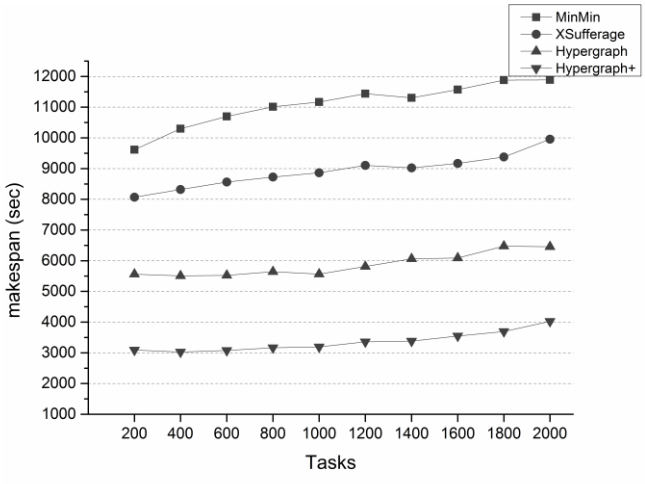
The metrics used for evaluating the scheduling algorithms are makespan, I/O reduction percentage, and running time. The makespan, i.e., the overall execution time, is the most common

performance measure for a scheduling algorithm. A lower makespan means better performance of the scheduling algorithm. The I/O reduction percentage was calculated as the ratio of the amount of data sets accessed from the local disk storage to the total amount of data sets required by the tasks. A higher I/O reduction percentage means a greater decrease in data transfers. The running time is the time spent scheduling tasks to computing processors, and reflects the time complexity of scheduling algorithms. A scheduling algorithm will be more efficient with less running time. All three metrics provide a complete evaluation for each scheduling algorithm.

In our experiments, the target application computed a digital elevation model of Gilmer County from the LiDAR dataset. The original point cloud was divided into different block sizes and this could lead to different task granularities and different degrees of I/O overlap, i.e., smaller block sizes created more I/O overlap. Illustrated from Figure 5 and the example in Section 3.2, the number of IDW interpolation tasks was equal to the number of point blocks. Experimental results are illustrated in Figures 10 and 11.



(**a**)



(**b**)

Figure 10. Performance evaluation with different numbers of tasks. (**a**) Makespan; (**b**) I/O reduction percentage.

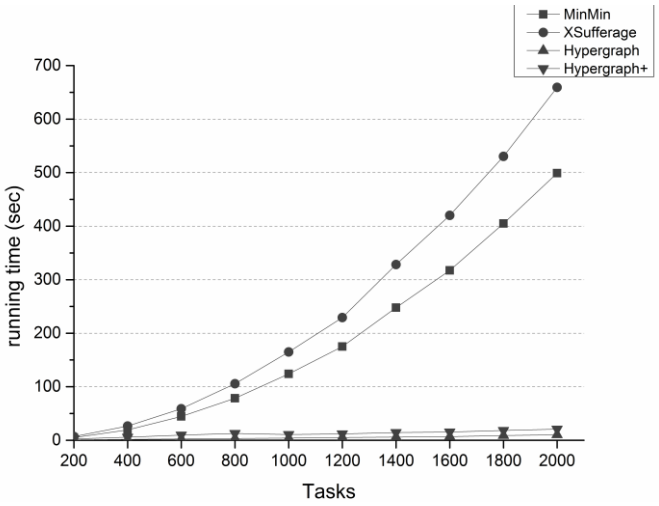


Figure 11. Efficiency evaluation with different numbers of tasks.

As shown in Figure 10a, when the number of tasks increased, the makespan of *MinMin* increased quite rapidly; *XSufferage* and *Hypergraph* followed the same pattern, but the makespan of *Hypergraph+* grew much more slowly. During the entire task execution process, our proposed *Hypergraph+* algorithm reduced the total execution time for *MinMin*, *XSufferage*, and *Hypergraph* by 70%, 62%, and 43%, respectively. These results demonstrate that *Hypergraph+* outperforms the other scheduling strategies.

Figure 10b shows that the percentage of I/O reduction in these heuristics varied with the number of tasks. When the number of tasks increased, the percentage I/O reduction in *MinMin* was about 40%, *XSufferage* was nearly 55%, and *Hypergraph* was above 80%, but *Hypergraph+* achieved a 2%–5% higher reduction than *Hypergraph*. In terms of the I/O reduction metric, *Hypergraph+* was superior to *MinMin*, *XSufferage* and *Hypergraph*.

As shown in Figure 10a,b, *MinMin* and *XSufferage* perform slower with lower I/O reduction than *Hypergraph+* and *Hypergraph*. This is because *MinMin* does not consider data sharing at all, and *XSufferage* fails to exploit data sharing patterns globally. *Hypergraph+* and *Hypergraph* take data sharing into consideration globally such that the tasks with shared input are assigned to the same processor as much as possible. In addition, *Hypergraph+* can obtain an optimal hypergraph partition result and maximizes the overlap probability between communication and computation to decrease the waiting time for tasks. Therefore, *Hypergraph+* achieves better performance than *Hypergraph* in terms of makespan and I/O reduction percentage.

As illustrated in Figure 11, when the number of tasks increased, the running time increased at a much faster rate for *MinMin* and *XSufferage*, in contrast to *Hypergraph+* and *Hypergraph*. *MinMin* and *XSufferage* must calculate the expected completion time for each task on each computing node to choose one task until all tasks are executed; consequently, the time complexity was $O(n^2)$. On the other hand, *Hypergraph+* and *Hypergraph* use hypergraph partitioning to map all tasks to processors quickly, as the time complexity of hypergraph partitioning was O(n) + O(logn) [31]. *Hypergraph+* was only about 3 s slower than *Hypergraph* on average. As seen in Figure 10a, *Hypergraph+* conserved more than 2400 seconds compared with *Hypergraph*. This small overhead can be negligible. Thus, *Hypergraph+* can achieve better performance than the other three algorithms and still maintains high efficiency.

6. Conclusions

This paper presents a *Hypergraph+* scheduling algorithm that extends the existing hypergraph-based scheduling algorithm for massive spatial data processing to obtain better performance. It first formulates a general hypergraph model to represent tasks, spatial datasets and processing platform. Then, the quality of hypergraph partitioning results is evaluated by a

Fitness function to map tasks to the processors such that the total volume of communication is minimized while balancing computational workloads. Moreover, *Hypergraph+* schedules tasks and file transfers to maximize the overlap probability between communication and computation with reduced end-point contention among processors. Simulations were carried out to compare *Hypergraph+* with *MinMin*, *XSufferage*, and *Hypergraph* using spatial interpolation applications on heterogeneous master-slave platforms. Simulation results illustrate that the *Hypergraph+* is on the average 43% better than *Hypergraph* in terms of makespan, while preserving the efficiency of *Hypergraph*.

In the future, we will extend the *Hypergraph+* algorithm to distributed file system storage centers. Currently, the distributed file system, e.g., Hadoop HDFS, is used to store and process massive spatial datasets. Data replication is often employed in Hadoop HDFS to improve availability and throughput. Therefore, our *Hypergraph+* scheduling algorithm can be further investigated to address the data replication problem and exploit a higher degree of data sharing in a Hadoop environment.

Acknowledgments: This work is supported by the Natural Science Foundation of China (Grant No.: 41301411) and the Natural Science Foundation of Hubei Province (Grant No.: 2015CFB399).

Author Contributions: Xuefeng Guan amd Bo Cheng conceived and designed the experiments; Bo Cheng performed the experiments; all the authors analyzed the data; Xuefeng Guan, Bo Cheng and Rui Li wrote the paper. Authorship must be limited to those who have contributed substantially to the work reported.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, Y.; Chen, B.; Yu, H.; Zhao, Y.; Huang, Z.; Fang, Y. Applying GPU and POSIX thread technologies in massive remote sensing image data processing. In Proceedings of the 19th International Conference on Geoinformatics 2011, Shanghai, China, 24–26 June 2011; pp. 1–6.
2. Song, W.; Yue, S.; Wang, L.; Zhang, W.; Liu, D. Task scheduling of massive spatial data processing across distributed data centers: What's new? In Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems, Tainan, Taiwan, 7–9 December 2011; pp. 976–981.
3. Xing, J.; Sieber, R.; Kalacska, M. The challenges of image segmentation in big remotely sensed imagery data. *Ann. GIS* **2014**, *20*, 233–244. [CrossRef]
4. Buyya, R.; Murshed, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CCPE* **2002**, *14*, 1175–1220. [CrossRef]
5. Maheswaran, M.; Ali, S.; Siegal, H.J.; Hensgen, D.; Freund, R.F. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99), San Juan, Puerto Rico, 12 April 1999; pp. 30–44.
6. Casanova, H.; Legrand, A.; Zagorodnov, D.; Berman, F. Heuristics for scheduling parameter sweep applications in grid environments. In Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), Cancun, Mexico, 1 May 2000; pp. 349–363.
7. Khanna, G.; Vydyanathan, N.; Kurc, T.; Catalyurek, U.; Wyckoff, P.; Saltz, J.; Sadayappan, P. A hypergraph partitioning based approach for scheduling of tasks with batch-shared I/O. In Proceedings of the 5th International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, UK, 9–12 May 2005; pp. 792–799.
8. Berge, C. *Graphs and Hypergraphs*; North-Holland Publishing Company: Amsterdam, The Netherlands, 1973.
9. Karypis, G.; Kumar, V. Multilevel k-way hypergraph partitioning. *VLSI Des.* **2000**, *11*, 285–300. [CrossRef]
10. Catalyürek, U.V.; Aykanat, C. PaToH (partitioning tool for hypergraphs). In *Encyclopedia of Parallel Computing 2011*; Springer Science & Business Media: Berlin, Germany, 2011; pp. 1479–1487.
11. Trifunovic, A.; Knottenbelt, W.J. Parkway 2.0: A parallel multilevel hypergraph partitioning tool. In Proceedings of 19th International Symposium on Computer and Information Sciences (ISCIS 2004), Kemer-Antalya, Turkey, 27–29 October 2004; pp. 789–800.
12. Alpert, C.J.; Kahng, A.B. Recent directions in netlist partitioning: A survey. *Integr. VLSI J.* **1995**, *19*, 1–81. [CrossRef]
13. Karypis, G.; Aggarwal, R.; Kumar, V.; Shekhar, S. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **1999**, *7*, 69–79. [CrossRef]

14. Mobasher, B.; Jain, N.; Han, E.H.; Srivastava, J. *Web Mining: Pattern Discovery from World Wide Web Transactions*; Technical Report TR96-050; Department of Computer Science, University of Minnesota: Minneapolis, MN, USA, 1996.
15. Demir, E.; Aykanat, C.; Cambazoglu, B.B. Clustering spatial networks for aggregate query processing: A hypergraph approach. *Inf. Syst.* **2008**, *33*, 1–17. [CrossRef]
16. Catalyürek, U.V.; Aykanat, C. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.* **1999**, *10*, 673–693. [CrossRef]
17. Cambazoglu, B.B.; Aykanat, C. Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids. *IEEE Trans. Parallel Distrib. Syst.* **2007**, *18*, 3–16. [CrossRef]
18. Kaya, K.; Aykanat, C. Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments. *IEEE Trans. Parallel Distrib. Syst.* **2006**, *17*, 883–896. [CrossRef]
19. Doumbouya, M.B.; Kamsu-Foguem, B.; Kenfack, H. Argumentation semantics and graph properties. *Inf. Process. Manag.* **2016**, *52*, 319–325. [CrossRef]
20. Kamsu-Foguem, B.; Tchuenté-Foguem, G.; Foguem, C. Conceptual graph operations for formal visual reasoning in the medical domain. *IRBM* **2014**, *35*, 262–270. [CrossRef]
21. Kamsu-Foguem, B. Knowledge-based support in Non-Destructive Testing for health monitoring of aircraft structures. *Adv. Eng. Inf.* **2012**, *26*, 859–869. [CrossRef]
22. Kosar, T.; Balman, M. A new paradigm: Data-aware scheduling in grid computing. *Futur. Gener. Comput. Syst.* **2009**, *25*, 406–413. [CrossRef]
23. Caíno-Lores, S.; Carretero, J. A survey on data-centric and data-aware techniques for large scale infrastructures. *World Acad. Sci. Eng. Technol. Int. J. Comput. Electr. Auto. Cont. Inf. Eng.* **2016**, *10*, 459–465.
24. Mohamed, H.H.; Epema, D.H. An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In Proceedings of the IEEE International Conference on Cluster Computing 2004, Los Alamitos, CA, USA, 20–23 September 2004.
25. Zhang, Y.F.; Tian, Y.C.; Fidge, C.; Kelly, W. Data-aware task scheduling for all-to-all comparison problems in heterogeneous distributed systems. *J. Parallel Distrib. Comput.* **2016**, *93*, 87–101. [CrossRef]
26. Szmajduch, M.; Kołodziej, J. Data-aware scheduling in massive heterogeneous systems. In Proceedings of the 29th European Conference on Modeling and Simulation ECMS 2015, Varna, Bulgaria, 26–29 May 2015; pp. 608–614.
27. da Silva, F.A.; Senger, H. Scalability limits of Bag-of-Tasks applications running on hierarchical platforms. *J. Parallel Distrib. Comput.* **2011**, *71*, 788–801. [CrossRef]
28. Guan, Q.; Clarke, K.C. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *Int. J. Geogr. Inf. Sci.* **2010**, *24*, 695–722. [CrossRef]
29. Muthuvelu, N.; Liu, J.; Soe, N.L.; Venugopal, S.; Sulistio, A.; Buyya, R. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In Proceedings of the 3rd Australasian workshop on Grid computing and e-Research (AusGrid 2005), Newcastle, Australia, 30 January–4 February 2005; pp. 41–48.
30. Guan, X.; Wu, H. Leveraging the power of multi-core platforms for large-scale geospatial data processing: Exemplified by generating DEM from massive LiDAR point clouds. *Comput. Geosci.* **2010**, *36*, 1276–1282. [CrossRef]
31. Trifunović, A.; Knottenbelt, W.J. Parallel multilevel algorithms for hypergraph partitioning. *J. Parallel Distrib. Comput.* **2008**, *68*, 563–581. [CrossRef]