Resource Allocation for Streaming Applications in Multiprocessors

Jacob Jan David Mol

September 17th, 2004

Delft University of Technology
Faculty of EEMCS

Royal Philips Electronics
Philips Research Eindhoven

Resource Allocation for Streaming Applications in Multiprocessors

Master's Thesis

Computer Science

Jacob Jan David Mol

September 17th, 2004

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Parallel and Distributed Systems group

Royal Philips Electronics
Philips Research Eindhoven
Information Technology group

Delft University of Technology
Faculty of EEMCS

Royal Philips Electronics
Philips Research Eindhoven

Graduation Data

Author	:	Jacob Jan David Mol
Title	:	Resource Allocation for Streaming Applications in Multiprocessors
Graduation Date	:	September 27th, 2004
Graduation Committee	:	
dr. ir. D.H.J. Epema		Delft University of Technology
prof. dr. ir. H.J. Sips		Delft University of Technology
dr. P.G. Kluit		Delft University of Technology
ir. O.M. Moreira		Royal Philips Electronics
dr. ir. M.J.G. Bekooij		Royal Philips Electronics

Abstract

A multiprocessor system which can run multiple hard real-time jobs consisting of multiple tasks simultaneously, has to be able to guarantee there will be no resource conflicts among the running tasks. This can be accomplished by allocating the required resources when a job is started. These jobs will be described as sets of functional components with data dependencies. These components are to be distributed over the processors, such that their timing constraints are not violated. Which resources to allocate depends on the state of the system and the job at hand, making this a problem which has to be solved at run time.

This report addresses this resource allocation problem for homogeneous systems consisting of 10–30 processors connected through a small network of 1–4 routers. The problem is tackled by using Bin Packing approximation algorithms as heuristics, augmented with techniques to keep the bandwidth usage to a minimum. The packing algorithms will try to map neighbouring components on the same processor. In addition, a clustering of the tasks of each job, which is optimised for bandwidth usage, will be calculated at design time. The packing algorithm will first try to map this pre-calculated solution, before it tries to map the original components.

This results in a significant reduction in resource usage, without increasing the chance a job cannot be placed. Empirical tests indicate that such a system is capable of using 95% of the available computational resources, during a period with a large number of task arrivals and departures on a heavily loaded system.

Preface

The Hijdra project at Philips Research is concerned with the creation of an environment to run both hard and soft real-time streaming media jobs consisting of multiple tasks on embedded multiprocessors. The problem of finding good strategies for the mapping of such jobs on such systems arose in this context. This report is the result of my MSc project on this subject, as part of the curriculum of Computer Science at Delft University of Technology. It addresses the mapping problem for hard real-time tasks on homogeneous systems.

I would like to thank my advisor at Philips, ir. O.M. Moreira, and the Hijdra project leader, dr. ir. M.J.G. Bekooij, for their help and support. From the side of the Delft University of Technology, I would like to thank ir. dr. D.H.J. Epema for his help and guidance to enhance the technical quality and readability of this report. Finally, I would like to thank Christie Maas for her support and patience during all stages of the project.

I wish you an interesting read,

Jan David Mol
Delft, September 17th, 2004

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The complexity of consumer electronics is ever increasing. Products such as mobile phones, DVD players, and portable MP3 players are required to process multiple media streams at the same time. In addition, the computational complexity of processing each of these streams is growing as well, demanding more computational power. For some streaming media applications, this must be achieved without causing a degradation in quality due to interference between the processing of the streams.

When the device can run more jobs than it can run simultaneously, the assignment of a job to the processors has to be determined at run time. A look-up table could be employed to determine these assignments for each change in the set of running jobs. However, such a table can grow quite large if more than a few jobs can run simultaneously, or if the assignment of a job to the processors depends on the assignment of the other jobs.

The Hijdra project at Philips Research aims to provide a more dynamic solution to this problem, without sacrificing performance. By designing a system which is predictable in its timings and resource availability, tight guarantees can be made for the jobs running on it. The Hijdra project designs such a system from the ground up, allowing the predictability of the hardware to guarantee the predictability for the software. Its focus lies on hosting streaming media jobs, both soft and hard real-time, in an embedded multiprocessor system. These jobs are started and stopped by a user on a system of approximately 10–30 possibly heterogeneous processors.

The jobs can consist of up to 100 tasks, each of which needs to be assigned to a processor when the application is requested to run. This assignment depends on the current state of the system and thus has to be performed at run time. Since finding a valid assignment for such jobs in general is an NP-complete problem, it is a challenge to find good or even valid mappings within a time span on the order of milliseconds.

This report will focus on the hard real-time tasks within this Hijdra framework, and will make an attempt to tackle this challenge for homogeneous systems. First, the hardware and software models used in Hijdra will be described in Chapter 2. Next, Chapter 3 describes the tasks of the Resource Manager, which is responsible for the brokering of resources in the system amongst the applications. The case of a system with its processors connected by a single router is analysed in Chapter 4. This is extended to the general case of allowing any network topology in Chapter 5. The final conclusions and future considerations are discussed in Chapter 6.

Chapter 2

Environment

The timing requirements of a hard real-time application are strict. If the system cannot guarantee the availability of enough resources, the job can miss its deadline. Since the job is hard real-time, this is catastrophic for the quality or even the application as a whole.

In the Hijdra architecture, the applications are divided into functional components with data dependencies which can be distributed over the processors in the system. For each component, the worst-case resource requirements are known at design time.

The timing and throughput characteristics of the application can be derived, by forcing these applications to conform to a strict model called Synchronous Data Flow (SDF). If these characteristics turn out to be unsatisfactory, the resource requirements can be stretched to allow better results.

When these components are mapped to the hardware, enough resources have to be reserved in the system at run time for each component. The timing and throughput characteristics derived by the analysis have to be guaranteed. For this to be possible, the resources allocated in the system have to be enough to cover any unpredictability. Hence, it is logical to use hardware which is predictable in all relevant aspects. The usage of predictable hardware decreases the gap of the amount of resources required by the application, and the amount that has to be reserved for it.

The next section will describe a template for such hardware. In Section 2.2, a template for the applications will be presented, as well as the tools necessary to analyse their behaviour. How the applications are executed on the hardware is described in Section 2.3.

2.1 Hardware Template

The hardware is a multi-processor system, consisting of a set of processing tiles connected by a network of routers. Every tile contains a single processor, its own local memory, and an interface to the network. The network itself consists of a set of routers, forming any topology using unidirectional links. An example of such a system is shown in Figure 2.1.

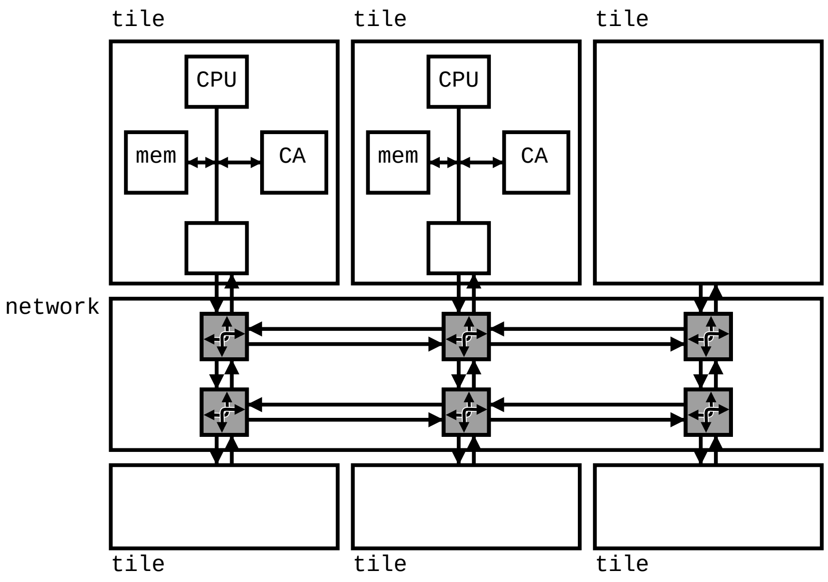

Figure 2.1: A network connecting six processing tiles.

The setup of the processing tiles will now be discussed, as well as the characteristics of the network. This network will for now be assumed to consist of a single router. The characteristics of a network involving multiple routers, such as the routing, will be discussed in Chapter 5, which deals with the analysis of systems employing more than one router.

2.1.1 Processing Tiles

A processing tile in a Hijdra system is an independent computational unit, consisting of:

- A main CPU, which executes the tasks assigned to the tile.
- A local memory, which holds the local state of the tasks as well as the packets of data which are to be transmitted or have been received over the network.
- A network interface (NI), which transmits and receives packets over the network. It takes care of the construction and decomposition of each packet into a header and a payload. To do so, it maintains a table of all the network connections to and from the tile.
- A communication assist (CA), which arbitrates the access to the memory for the CPU and the NI.

These components are connected as in Figure 2.2. For both the CPU and the NI, the memory should act in a predictable way. This implies that the bandwidth towards the memory should be divided between them, such that the stalls on the reads or writes from either side can be tightly bounded. The CA facilitates this arbitration, and can for example allow the NI and CPU access to the memory ever other cycle. Such a scheme would delay any read or write by at most one clock cycle.

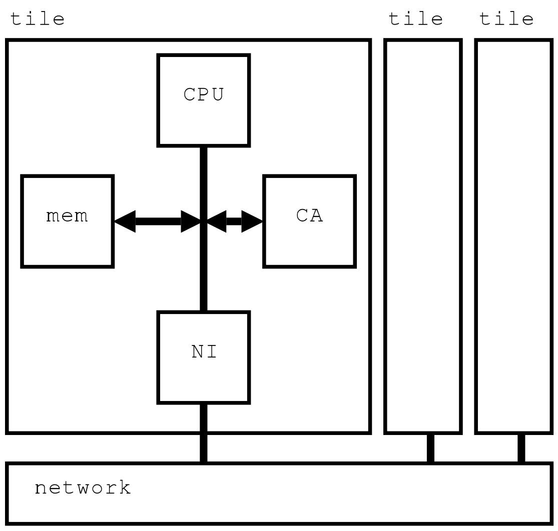

Figure 2.2: A processing tile.

2.1.2 Network

The network in the Hijdra architecture can have any topology. The links in the network can connect routers to each other or to processing tiles, using unidirectional links. Every link provides the same amount of bandwidth, but multiple links are allowed to exist between any two nodes. Typically, connected nodes have links between them in both directions. An example of this can be seen in Figure 2.1.

A router is capable of transferring data between all inputs and all outputs simultaneously. The routers are synchronised and act on a time slice basis, copying data from all used inputs to all used outputs in every time slice. Because the routers are predictable in their timings, the system can avoid collisions within these routers by reserving time slices for every connection. In effect, the routers preserve the order in which they receive their data, making them capable of serving FIFO channels.

Assuming collisions are avoided, the routers form no bottleneck in the network since they can handle any load the links provide. The throughput of the system is therefore bounded by the bandwidth provided by the links.

If the processing tiles are all connected to a single router, forming a star topology, then the bandwidth available between any two tiles is limited only by the available outgoing bandwidth of the source and the available incoming bandwidth of the sink.

Such a topology would be ideal, but cannot scale. The routers use a full interconnection network internally, which grows quadratically in the number of links. Due to physical constraints, a single router can be connected to at most 10 input and 10 outputs. A system which contains only a single router can therefor contain at most 10 processing tiles.

The routers thus described are modelled after those described in the Æthereal model [8], which is internally being developed at Philips Research. It describes a network-on-chip, and is capable of providing the necessary guaranteed throughput connections. They will be further described in Chapter 5, in which topologies of multiple routers are considered. The issues of route discovery and allocation will be dealt with there as well.

2.2 Software Template

The Hijdra architecture is designed to run real-time, streaming media applications. It describes these applications by a three layer system. Figure 2.3 is an example of this layering. The top layer is the application, which represents the collection of all the programs offered by the system.

Such an application consists of several jobs, which form the middle layer. These jobs represent the individual programs which can be started or stopped at the user's request. More than one job can be running on the system at any given time.

The structure of a job can be represented by a directed graph. The bottom layer consists of the actors and channels which form these graphs. These actors are functional components, which communicate to each other solely through fixed, unidirectional, point-to-point connections.

Since the jobs to run are hard real-time, the resource requirements of every actor has to be known at design time. The requirements imposed by a channel are less straightforward, since they depend on the amount of data it transports, as well as the maximum allowed latency for this data.

To derive these requirements, the Hijdra architecture uses SDF graphs to model the jobs. These SDF graphs, standing for Synchronous Data Flow, are a model well known in the literature to model and analyse streaming media application [1, 12]. From the SDF graph, it is possible to derive the minimum guaranteeable throughput from input to output, as well as the required buffer sizes and bandwidth of each channel.

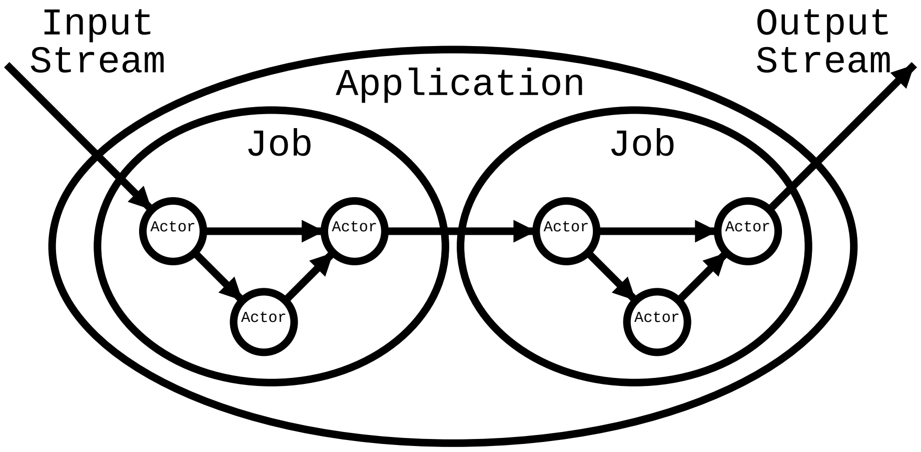

Figure 2.3: The three levels of granularity of an application.

2.2.1 SDF Graphs

An SDF graph is a directed graph $G(V, E)$ consisting of a set of actors V and a set of channels E. The channels transport data in the form of *tokens*, which have a fixed size per channel. The channels are guaranteed to preserve the FIFO ordering of the tokens it transports.

The actors can communicate with each other through these channels only. If two actors are connected by a channel, the producer will produce a fixed number of tokens at every execution, and the consumer will consume a fixed number of tokens at every execution. These numbers are annotated on the graph, as can be seen in Figure 2.4. In this example, the dot on the channel from A to B represents an initial token to be present in that channel. It serves as a delay between A and B, because due to this token, the sample produced by execution t of A will be processed by execution $t+1$ of B.

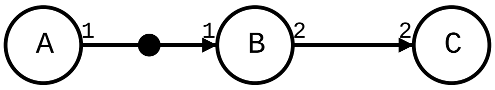

Figure 2.4: An SDF graph consisting of three actors.

An actor of a job is allowed to execute as soon as the annotated numbers of tokens are available on all its incoming connections. Once these conditions are met, the actor is said to *fire*. After execution, the actor will have consumed its input and produced its output.

This definition allows different actors to operate on different data samples at the same time. For instance in Figure 2.4, once actor A has produced output, actor B can consume it while actor A starts to work on producing a new sample. In fact, if enough input is available, an actor can fire multiple times simultaneously. If it does, it is required to maintain the FIFO ordering of the data stream.

HSDF Graphs

From this information, several essential properties can be derived. This is done after converting the SDF graph into a simpler, homogeneous SDF (HSDF) graph. In such a graph, every actor produces or consumes exactly one token on each of the channels it is connected to. Such a graph is easier to analyse due to its direct correspondence to normal graph theory. The transformation of SDF to HSDF can be done relatively easy (by cloning actors), but this can result in an exponentially larger graph.

2.2.2 Throughput Analysis

In theory, actor A in Figure 2.4 can fire infinitely often at once. In practice, it is assumed the executions of A are controlled by an external source, such as a stream from disk or from a recording device, as shown in Figure 2.3. The firing rate of the actors is also limited by any data that has to be communicated to the processing of the next data sample. This can occur for example in a video stream, where frames are incrementally updated, and thus making the content of one frame depend on the content of the previous one.

Cycles

This type of behaviour can be modelled by adding cycles to the SDF graph, for instance as in Figure 2.5. For the first firing of actor B, there exists no previous data sample. But because an actor can only fire once all its input channels contain enough tokens, the model requires tokens to be present on the back edge in the cycle to accommodate for the first firing. After B has executed, C can work on B's output and produce information for B to use on the next sample. The execution of C thus delays the next execution of B.

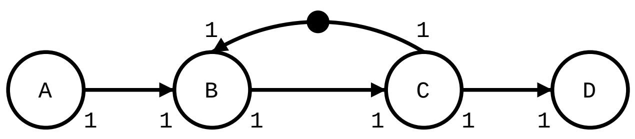

Figure 2.5: An SDF graph containing a cycle.

Preserving Internal State

It often occurs that an actor has to preserve information for itself between executions. This internal state of an actor is modelled through adding a self edge to the actor, as in Figure 2.6. Since this self edge is in essence a small cycle, it has to contain a token, which represents the initial internal state.

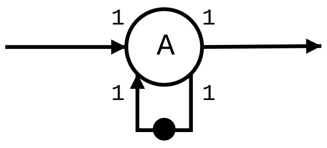

Figure 2.6: An SDF actor with a self edge.

Execution Times

If there are no cycles in the SDF graph, the minimum guaranteeable throughput is infinite, because all actors can fire on multiple samples at once, without waiting for the computations of previous cycles to be completed.

By introducing cycles, the firing of an actor is potentially delayed due to waiting for another actor to finish execution on the previous sample. If the cycle is a self-edge, the actor has to wait for itself to finish processing the previous sample. The execution times of the actors can thus influence the throughput of the system.

Because the tasks under consideration are hard real-time, their worst-case execution times are known. From this, the throughput through each cycle in the SDF can be calculated.

Cycle Means

The minimal throughput that can be guaranteed to flow through the SDF is limited by the cycles in the graph. Given an HSDF graph $G(V, E)$, then for every cycle $C = \{v_1, e_1, v_2, e_2, \ldots, v_n\}, v_1 = v_n$, the cycle mean μ_C can be calculated. This cycle mean denotes the average time it takes to execute a full cycle, given the worst-case execution times of its actors and the number of initial tokens (delays) on its channels. Let $d(e)$ be the number of delays on channel e, $W(v)$ the worst-case execution time of actor v and $\mathcal{C}(G)$ be all the cycles in G, then:

$$\mu_C \quad := \quad \frac{\sum_{i=1}^{n-1} W(v_i)}{\sum_{i=1}^{n-1} d(e_i)}.$$

The minimal guaranteeable throughput is limited by the highest μ_C present in the HSDF. So, let $\mathcal{C}(G)$ be all cycles in G. Then the Maximum Cycle Mean (MCM) μ_G is equal to

$$\mu_G \quad := \quad \max_{C \in \mathcal{C}(G)} \mu_C.$$

Which results in a minimal sustainable throughput of

$$\Theta \quad = \quad \frac{1}{\mu_G}.$$

This maximum sustainable throughput is not always what is required by the application, in which case we can allow a less strict system, by stating the desired sustainable throughput θ, and deriving the maximum allowed cycle mean μ_θ:

$$\frac{1}{\mu_\theta} \quad := \quad \theta \leq \Theta.$$

A *critical cycle* is one which has a mean equal to μ_θ.

2.2.3 Channel Capacities

The unlimited capacity of the channels in an SDF graph is something which cannot be implemented in hardware. Through analysis, the maximal required channel sizes can be derived. The calculated bounds can then be represented in the HSDF model by adding back edges containing a number of tokens equal to this bound. For example, Figure 2.7 shows a channel from producer P to consumer C with a capacity of two tokens. Between the two edges, there can never be more than two tokens in total, bounding the original forward edge in capacity.

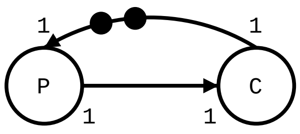

Figure 2.7: A channel from P to C with a back edge added to bound its capacity.

Because these back edges form extra cycles in the graph, they can influence the throughput. Take for example the HSDF graph drawn in Figure

2.8a. For clarity purpose, the token production/consumption numbers are not drawn, but are all 1. All three actors have the same worst-case execution time W. As the only cycles in the graph are formed by an actor and its self edge, the MCM of the graph is $\mu_G = W$. Suppose $\mu_\theta = \mu_G$. Now, the channels will be bounded in their capacities. A first attempt results in Figure 2.8b. For each channel, the back edge creates a cycle C which has to have $\mu_C \leq \mu_\theta = W$. Since such a cycle C contains two actors, with $2W$ execution time in total, it has to contain at least two tokens to obtain a $\mu_C \leq \frac{2W}{2} = W$.

Figure 2.8b shows this is not all there is to it. The addition of channel back edges have created extra cycles in the graph. In particular, take the cycle drawn in gray. It contains all three actors, but only two tokens. Therefor, the mean of this cycle is equal to $\frac{3W}{2} > \mu_\theta$, making the graph unable to guarantee the desired throughput. To fix this, another token has to be added to this cycle. Only channel back edges are candidates for such extra tokens, since adding tokens to the original edges from Figure 2.8a would change the semantics of the graph. A token is therefor added to the edge (B, A), resulting in Figure 2.8c. In this final graph, no cycle has a mean larger than μ_θ.

This example shows that the capacity of a channel cannot be derived purely by the characteristics of its endpoints. Adding back edges to the system can form additional cycles, making it necessary to reanalyse the throughput after the back edges are added. The impact on the throughput of bounding the channel sizes can then be derived using the same methods as before. This allows verification of the throughput characteristics once all channels have been bounded in size.

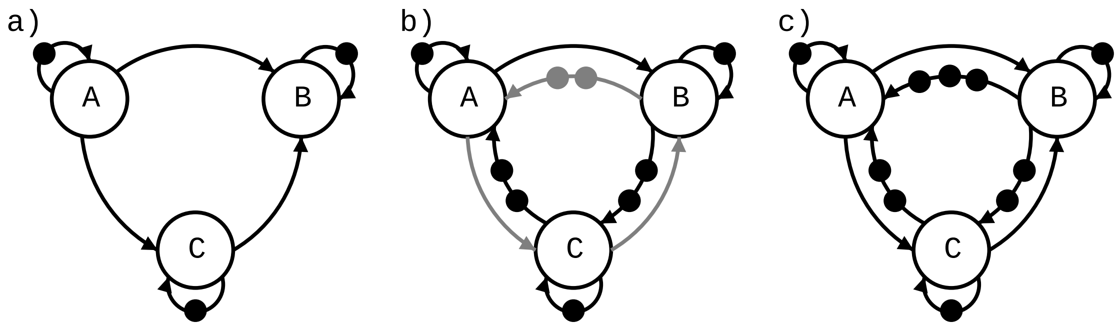

Figure 2.8: a) An HSDF graph with 3 actors. b) The same graph with FIFO back edges added. One of the extra cycles that has emerged is drawn in gray. c) The same graph, with a token added to (B, A).

2.3 Execution Model

The main CPU on a processing tile keeps a list L of tasks which are assigned to run on it. Since only periodic tasks are considered, this list does not change until tasks are added or removed by external events.

The CPU will work through the list by inspecting the tasks in a round-robin fashion. If the task at hand is ready to execute, the CPU will do so until the task is finished. Whether or not the task could execute or not, the next task is considered, without changing the order of the tasks in the list. The following pseudo code reflects this process:

```
while true
  for i := 1 to |L|
    if can_execute( L[i] )
      execute( L[i] )
    end if
  end for
end while
```

in which the `can_execute(t)` function returns `true` iff all input for task `t` is available on its incoming edges. Because of how FIFO buffers are modelled, this also ensures enough space is available to write its output on its outgoing edges.

This implies that any task which becomes ready for execution, has a maximum waiting time equal to the sum of the execution times of all the other tasks in the list. The worst-case execution time of every task is known at design time. If $W(v)$ is the worst-case execution time of task v, and D_i is its deadline within a single period, then the following simple formula ensures every task on the processor will meet its deadline:

$$\forall i \in L : D_i \quad \geq \quad \sum_{v \in L} W(v).$$

The software model allows the determination of deadlines for the actors, such that a desired temporal behaviour can be guaranteed, independent of the mapping of tasks on processors, or the ordering of tasks in L.

2.3.1 Network Latency

The presence of a network influences the behaviour of the job's execution. When an SDF channel has its endpoints mapped on different processing tiles, the network acts as an added latency for each data transfer. This can be modelled by adding an extra actor between the endpoints of the channel, which copies data from input to output taking the given latency as its execution time. If such a channel is part of a cycle, this increases the mean of that cycle.

To make the mapping of a job to processing tiles easier, it will be assumed at design time that every channel can potentially be mapped over the network, at maximum distance from each other. This increases the mean of all cycles in the graph. If this is unacceptable, some or all actors in the cycle will be merged into a single actor, such that they will be mapped to the same processing tile. In the extreme case, the graph will have to be transformed into a directed graph of strongly connected components, which is acyclic except for the self edges. In such a graph, the network still adds latency to the graph, but not to any cycle.

Chapter 3

Resource Management

The processing tiles and the network links are inherently limited in the amount of resources they offer to applications. The Resource Manager (RM) is one of the pieces of middle-ware running in the Hijdra system, and is responsible for the brokering of these resources amongst the jobs which are running and arriving on the system.

The RM has to perform its job on-line: it has to map (and unmap) applications depending on the current state of the system. It has only limited resources available for this job. It has to be able to run on an ARM 200MHz processor and finish its computations in the order of tens of milliseconds on average. This places severe limitations on the possible computations which can be performed.

3.1 Job life-cycle

It is assumed that a request to start a job originates from an unpredictable source, such as a user or an external system. The RM has to act when either a job is started or stopped.

3.1.1 Starting Jobs

Once such a request arrives at the RM, the RM checks if a suitable assignment of actors to processors can be found, as well as routes through the network to support this mapping. Such an assignment is suitable, if it is valid. An assignment is valid if all the actors and routes actually fit into the parts of the system to which they are assigned. If the state of the system allows many valid mappings which are easy to find, the RM can do more: it can find a mapping such that jobs arriving in the future will have a higher chance of being mapped as well.

In the case that no valid mapping can be found, the job cannot be started, a fact which is then propagated back to the source that requested this. All this has to be decided in a short time span, on low speed CPUs. The user cannot be delayed for too long before the job is actually started or the start request is denied. The denial of start requests may also be handled by a Quality of Service manager, which can then try to start the job in a lower quality mode such that it needs less resources.

3.1.2 Stopping Jobs

A job can either finish or be requested to stop by an external source. From the viewpoint of the RM, the work to be done once a job is ready to be removed from the system, is fairly trivial: the resources used in the processing tiles it occupies have to be deallocated, as well as the routes through the network used by the job.

3.1.3 Resource Fragmentation

When the load in the system becomes high, new jobs requested to be started have a higher chance of becoming scattered over the processing tiles, to use up the resource leftovers. This implies that systems which frequently experience a high load have a high chance of having their resource usage fragmented. Since scattered jobs generally use more resources (such as network bandwidth) than ones which are mapped to a concentrated set of tiles, this leads to degeneration of the system. This fragmentation occurs at various resource types, including:

- *Actor scattering* due to the forced distribution of actors when starting jobs on heavily loaded systems.
- *Memory fragmentation* due to actors using memory regions on a tile which are not adjacent to each other or the memory boundaries.
- *Network degeneration* due to the increasing length of optimal routes once the network becomes congested.

3.2 Resource Modelling

The embedded multiprocessor template, as defined in the previous chapter, distinguishes several resources to be used by the jobs which are running. The exact requirements of a job in terms of these resources depends on the distribution of its actors over the available processors, which can be formalised for both the actors and the communication channels of a job.

The resources required of a processing tile will be represented as a vector, since a tile hosts several resources:

CPU cycles are offered in an endless supply, but because all applications running on it are periodic of nature, it cannot host an infinite number of actors. The analysis of SDF graphs provides a solution to this problem: as states in equation 2.1, an actor can only be added to a tile if the sum of all the execution times does not exceed any of the actors their deadline. The deadline of all actors in a job, which is requested to be started, is the same. From this information, an upper-bound on the available CPU space can be derived for every processor. Say L is the list of actors currently mapped to a processor p, and μ_θ is the deadline of the actors of the job to be started. Then the amount of available CPU space on that processor is equal to $\min(\mu_\theta, \min_{i \in L} D_i)$. If there are no actors mapped to a tile, it has an infinite amount of free CPU space.

Memory is offered by the tile to store data and stack.

NI FIFO slots need to be allocated for each slot used in the NI route table when a route over the network needs to be established.

CA bandwidth needs to be allocated for each network connection to allow the packets to be read or written to the memory.

Incoming bandwidth has an upper-bound by the number of incoming physical connections, and thus needs to be allocated.

Outgoing bandwidth has the same type of restriction as the incoming bandwidth, and thus needs to be allocated as well.

Given a set of processing tiles T, then for every tile $t \in T$, a vector of free resources $F(t)$ is known. It contains the amount of resources available in the order as described in the list above:

$$F(t) := \begin{bmatrix} C(t) \\ M(t) \\ N(t) \\ S(t) \\ I(t) \\ O(t) \end{bmatrix},$$

where $C(t)$ to $O(t)$ are the amounts of resources available on tile t. These vectors are adjusted every time a job is started or stopped.

Since latency is no issue, the network offers a sole resource: bandwidth between all the tiles. For the time being, the network shall be considered to consist of a single router. In this case, a scalar B is sufficient to represent the amount of bandwidth available in this router. In Chapter 5, the full network model shall be introduced and modelled.

3.2.1 Actors

An SDF actor represents a computation, and thus requires CPU cycles to run as well as memory to store the intermediate results. For actors that have a self-edge to model their state between periods, the memory to store this state is included in the memory requirements of the actor. This can be done without loss of generality, since the self-edge is always mapped to the same tile as its actor.

The requirements can be modelled as a function of vectors $s : V \to \mathbb{N}^d$:

$$s(v) := \begin{bmatrix} W(v) \\ M(v) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

in which $W(v)$ is the worst-case execution time of actor v, and $M(v)$ the required amount of memory.

3.2.2 Channels

The requirements of an SDF communication channel are more complex because the resources required by it depend on the location of the producing and consuming actor. For instance, if both are mapped onto the same processing tile, the communication channel can be implemented by a FIFO buffer in memory. Such a channel will be called *internally* mapped. On the other hand, if both ends are mapped onto different tiles, network resources are needed to transport the data from one tile to the other. Such a channel will be called *externally* mapped.

Internally mapped

If both endpoints of a channel are mapped to the same tile, it only requires memory on that tile to store the FIFO buffer. Written as a vector function, these requirements are modelled by a function of vectors $i : E \to \mathbb{N}^d$:

$$i(e) := \begin{bmatrix} 0 \\ M(e) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

in which $M(e)$ is the amount of memory required to store the FIFO buffer for channel e if both its endpoints run on the same processing tile.

Externally mapped

If the endpoints of a channel are mapped to distinct tiles, a link over the network has to be established to support it. The following resources are required from the tiles hosting the producer and the consumer of the channel:

- *Memory*, on both tiles, to store the send and receive buffers.
- A *NI slot*, on both tiles, to register a connection in the network interface.
- *Outgoing bandwidth* on the tile hosting the producing actor.
- *Incoming bandwidth* on the tile hosting the consuming actor.

$$p(e) := \begin{bmatrix} 0 \\ M_p(e) \\ 1 \\ CA_p(e) \\ 0 \\ b(e) \end{bmatrix}, \qquad c(e) = \begin{bmatrix} 0 \\ M_c(e) \\ 1 \\ CA_c(e) \\ b(e) \\ 0 \end{bmatrix}.$$

In the above equations, $b(e)$ is the amount of bandwidth required by the channel, $M_p(e)$ and $M_c(e)$ the amount of memory required to store the producing and the consuming end, respectively. The required CA bandwidth is represented by $CA_p(e)$ and $CA_c(e)$. A single NI FIFO slot is needed on both sides as well.

In the network, a route is required providing the bandwidth between both tiles. The amount of bandwidth required by the channel is represented by the scalar function $b : E \to \mathbb{N}$, which is equal to rate at which tokens are produced, times the token size.

3.2.3 Reformulation

In the previous sections, four different classes of size vectors have been defined which together describe the resource requirements of a job in a Hijdra system. For mapping purposes, this classification contains redundancy which can be removed to create a simpler, but idempotent description. Let $w(v)$ be the sum of

- the size of actor v, $s(v)$,
- for all channels e, in which v is the producer, $p(e)$,
- and for all channels e, in which v is the consumer, $c(e)$.

In other words, $w(v)$ represents the amount of resources needed to host actor v, if no other actors would be mapped to that tile. This can be calculated through

$$w(v) \quad := \quad s(v) + \sum_{e=(v,b)\in E} p(e) + \sum_{e=(b,v)\in E} c(e).$$

This is the size of an actor if all channels are externally mapped. On the other hand, if a channel is internally mapped, it has a different size. This gain $\delta(e)$ can be calculated through

$$\delta(e) \quad := \quad p(e) + c(e) - i(e).$$

Note that it is reasonable to assume that

$$\forall e \in E : p(e) + c(e) \quad \geq \quad i(e),$$

since storing both endpoints on the same tile reduces the amount of resources used. This implies that $\delta(e) \geq 0$.

The functions w and δ thus describe the resource requirements of the actors, and the gain of placing adjacent actors on the same processor, respectively. This information is sufficient to calculate the requirements of any mapping.

Chapter 4

Single Router Networks

The full on-line resource allocation problem is complex. As will be shown, it is NP-complete as are many of its subproblems. This fact makes finding optimal solutions impractical for anything but small applications, especially in a run time setting.

The approach taken to solve this is by splitting it up into a sequence of subproblems, for which approximations and heuristics are explored. The first step in this is to start with analysing small systems, in which all processing tiles can be connected to a single router. A router can be connected to up to 10 processing tiles, making this a realistic case for small systems.

Since a router can copy data from all inputs to all designated outputs simultaneously, no congestion or collisions can occur. The bottlenecks for the traffic travelling through the router is therefore always at the processing tiles. This allows the incoming and outgoing bandwidth to be modelled as resources to be bound as dimensions next to CPU and memory usage.

4.1 Problem Definition

When the network consists of a single router, there is no need to look for routes when the source and the sink of a channel are mapped on different tiles. This reduces the resource allocation problem to looking for a processing tile to host each actor. The resources available at the processing tile include the bottleneck between it and the router. The problem is thus defined:

Problem 1 (Single Router Allocation Problem). *Given a job digraph $G = (V, E)$ and a set of tiles T, in which every tile has $F : T \to \mathbb{N}^d$ resources still available and every actor has size $w : V \to \mathbb{N}^d$. Every channel reduces its endpoint resource requirements by $\delta : E \to \mathbb{N}^d$ if mapped internally. Does there exist a mapping $map : V \to T$, such that*

$$\forall t \in T : \sum_{\substack{v \in V \\ map(v)=t}} w(v) - \sum_{\substack{e=(v,w)\in E \\ map(v)=map(w)=t}} \delta(e) \quad \leq \quad F(t)$$

holds?

In other words, given a job digraph and a set of tiles, it is required to find a mapping, taking into account the maximum available space per tile in each resource dimension.

4.2 Optimality

The problem as described only considers whether a feasible solution exists. In many situations, for instance for an empty system, there exist an abundant number of possible solutions. This allows looking for optimality criteria.

On the one hand, power consumption can be a significant factor in embedded systems. In this single router case, the router does not impose feasibility constraints, since it can support full bandwidth between any pair of endpoints. But transferring data from NI, through the router, and to another NI costs power, which can be saved if the channel endpoints would be mapped to the same tile. From a power consumption perspective, it is thus preferable to use as little bandwidth as possible. If the system is capable of shutting down unused processors, it is also beneficiary in terms of power consumption to use as few processing tiles as possible.

On the other hand, the mapping of a job influences the possibilities for future jobs to be mapped. The system cannot know which jobs will arrive in the future, but it can predict which type of configurations it likes to avoid. If the available resources are scattered throughout the system, it becomes hard or even impossible to map actors with a high resource demand. A tighter packing can be obtained by using as few processing tiles as possible, thus keeping the number of unused tiles at a maximum. More free space can be obtained by using as little bandwidth as possible, since mapping two endpoints of a channel on the same tile saves on the available channels to and from that tile.

Because there are essentially two optimisation criteria, bandwidth and tile usage, there need not be a single global optimum. Deciding which criterium is more important than the other depends on the situation at hand. If the available bandwidth is running low, it is important to leave as much as possible for jobs to come. If there is plenty of bandwidth available, but the number of free tiles becomes critical, it is better to optimise for tile usage first.

A trade-off is required between the bandwidth used, and which tiles are used to host the job. The amount of bandwidth used is kept to a minimum by mapping as many channels internally as possible. For a job which does not share any processing tiles with other jobs, the fewer tiles it uses, the better.

If there exist partially occupied processing tiles within the system, it could be preferable for an arriving job to try and fill up such tiles first. In the trade-off, this can be represented by giving such tiles a lower cost to use.

4.3 Integer Programming Model

There is now enough information to construct an Integer Linear Program for the single router case. This program will be explained, and then simplified to allow approximation algorithms to be devised.

The required mapping function $map : V \rightarrow T$ is constructed through an enumeration of both the actors and the tiles, along with a matrix X of size $|T| \times |V|$, for which:

$$x_{ij} = \begin{cases} 1 & \text{iff actor } j \text{ is mapped to tile } i. \\ 0 & \text{otherwise.} \end{cases}$$

First, we need to make sure that every actor will be put in exactly one tile:

$$\begin{aligned} j = 1, \dots, |V| : \quad & \textstyle\sum_{i=1}^{|T|} x_{ij} = 1 \\ i = 1, \dots, |T|, j = 1, \dots, |V| : \quad & x_{ij} \in \{0, 1\}. \end{aligned}$$

To know where the edges are, a set of auxiliary variables r_{ei} is introduced, for which:

$$r_{ei} = \begin{cases} 1 & \text{iff channel } e \text{ is mapped internally on tile } i. \\ 0 & \text{otherwise.} \end{cases}$$

If the variables r_{ei} are maximised in the cost function, the following formulas can be used to program them:

$$\begin{aligned} i = 1, \dots, |T|, \forall e = (a, b) \in E : \quad & \tfrac{1}{2} x_{ia} + \tfrac{1}{2} x_{ib} \geq r_{ei} \\ i = 1, \dots, |T|, \forall e \in E : \quad & r_{ei} \in \{0, 1\}. \end{aligned}$$

Also, for each tile, the actors mapped to it should not need more resources than available. Each actor a requires $w(a)$ resources, with a bonus reduction of $\delta(e)$ for every edge that has both its endpoints mapped to the same processing tile:

$$i = 1, \dots, |T| : \quad \textstyle\sum_{j=1}^{|V|} w(v_i) x_{ij} - \sum_{e \in E} \delta(e) r_{ei} \leq F(t_i).$$

To be able to calculate the number of tiles used, an auxiliary variable u_i is introduced for each tile t_i, for which:

$$u_i \quad = \quad \begin{cases} 1 & \text{iff tile } i \text{ is used.} \\ 0 & \text{otherwise.} \end{cases}$$

If the variables u_i are minimised in the cost function, they can be programmed through:

$$\begin{array}{lll} i = 1, \ldots, |T| : & \sum_{j=1}^{|V|} x_{ij} & \leq |V| u_i \\ i = 1, \ldots, |T| : & u_i \in \{0, 1\}. & \end{array}$$

the bandwidth usage versus tile usage tradeoff will be represented by a vector $c_i, i = 1, \ldots |T|$ which represents the costs for using each tile. The optimal value for this trade-off can then be found through the objective function

$$\max \quad \alpha \sum_i \sum_{e \in E} r_{ei} b(e) - \sum_{i=1}^{|T|} c_i u_i \quad . \tag{4.1}$$

4.4 Approximation

The Integer Program defined in the previous section cannot be solved at run time. A heuristic is called for, and can be found by drawing analogies to known problems in the literature. The single-router case shows resemblance to a well-known problem in the literature called Bin Packing, which is defined as follows:

Problem 2 (Bin Packing Problem). *Given a set of items A with each $a \in A$ having a size $s(a) \in \mathbb{N}$, a bin size $S \in \mathbb{N}$, and a number $k \in \mathbb{N}$. Does there exist a partitioning of A into at most k subsets, such that for each subset P, $\sum_{x \in P} s(x) \leq S$ holds?*

This problem is NP-complete [11]. In our case, we want to pack actors into processing tiles. The size of an actor is equal to the amount of resources it requires. But because these requirements span multiple dimensions (such as CPU cycles and memory), this size cannot be represented as a scalar. In the literature, the problem of packing items with a vector instead of scalar size is known as Vector Packing, which is defined as follows:

Problem 3 (Vector Packing Problem). *Given a set of items A, with each $a \in A$ having a size $s(a) \in \mathbb{N}^d$, with $\forall i = 1, \ldots, d : s(i) \geq 0$, a bin size $S \in \mathbb{N}^d$ and a number $k \in \mathbb{N}$. Does there exist a partitioning of A into at most k subsets, such that for each subset P, $\sum_{x \in P} s(x) \leq S$ holds?*

Furthermore, our 'bins' are not always empty when a packing is performed: jobs may already be running in the system, partially filling various bins. The actors of these jobs cannot be moved to other tiles. Rather than solving a new Vector Packing problem each time a new job arrives, an existing packing is incremented. The extreme case of this is commonly called *on-line* packing: items are packed as they arrive in the system, without any knowledge of subsequent items and without the possibility to move already packed items. The case in which all items to be packed are known in advance is called *off-line* packing. In the resource allocation problem, the actors arrive in batches, thus falling between the on-line and off-line variants.

Another technical problem arises from the fact that the sizes of the objects to pack (the actors and their edges) are not constant: if two connected actors a and b are packed into the same tile, their combined size is $s(a) + s(b) - \delta((a, b))$. To be able to use known Vector Packing algorithms, this can be solved by assuming $\forall e \in E : \delta(e) = 0$, which effectively over-dimensions the problem.

Finally, the Vector Packing problem does not take bandwidth into account: it is only concerned about the number of bins used. This problem will be solved through trying to get heavily connected sets of actors on the same tile, but first, known algorithms for Vector Packing are explored and their suitability empirically tested. A more elaborate summary of these and other approximation algorithms for Vector Packing can be found in [14].

4.4.1 Off-line Vector Packing

For the off-line version of Vector Packing, algorithms are known to both solve the problem exactly and to approximate it. For an exact solution, the most common approach is to employ a branch-and-bound algorithm. Spread across literature there are several known lower-bound functions which are designed for, or can be adapted to Vector Packing [3, 13]. When these are combined and put into a branch-and-bound framework, an algorithm can be produced which can solve moderately sized (up to ~ 40 actors) instances on fast machines. It is doubtful whether it can be sped up to be able to solve most 100-actor instances on a 200 MHz ARM processor in a fraction of a second. This makes the calculation of exact solutions feasible only for design time solution generation purposes on moderately sized instances.

When turning to approximation algorithms, it should first be noted that it is impossible to approximate the number of required bins within an arbitrary constant (if P $\neq$ NP) [15]. This explains why these algorithms have a quite poor theoretical worst-case performance. The following algorithms are the most well-known. In all cases, d stands for the dimension of the vectors and bins, and OPT is number of bins required by the optimal solution:

First Fit (FF) For every item, put it in the bin in which it fits which has the lowest index. In the worst case, $(d + 7/10)OPT$ bins are required.

First Fit Decreasing (FFD) The items are sorted in non-increasing order after which First Fit is applied. There are several ways to order multidimensional items. If they are sorted with respect to the largest element, this algorithm requires at most $(d + 1/3)OPT$ bins.

FVL Fernandez de la Vega and Lueker provide an algorithm which requires at most $(d + \epsilon)OPT$ bins for arbitrarily small ϵ [7]. It is mostly of theoretical interest due to the fact that it is exponential in ϵ. For instance, it requires a running time of $O(n^{36})$ for $\epsilon = 1/3$.

CK Chekuri and Khanna use an approximation through linear programming, and re-pack the fractionally packed items using greedy methods [4]. They thus obtain an algorithm which requires at most $(2.58 + \ln(d))OPT$ bins.

The CK algorithm does not look suitable to be performed on-line, because it requires solving a large linear program. The FVL algorithm is unsuitable directly due to its high-order polynomial complexity. The FF and FFD algorithms therefore receive the most attention in this text.

First Fit

For the FF and FFD algorithms, several variations are known. A promising variant is BF and BFD, *Best Fit (Decreasing)*, which for each item, puts it in the bin in which it leaves the least free space. Since such variants require measuring the free space or item sizes of a vector, for which there is no unique way, they have not received any attention in the literature (to our knowledge).

Another variant of FF and FFD can be found in the way their packings can be computed. Instead of considering each item at a time and putting it into the first bin in which it fits, the exact same packing can be obtained by considering each bin at a time, trying to add every unpacked item to it, and do so if it fits. Although this does not change the resulting packing, it does not require all bins to remain 'open': once a bin is filled, no more items will be added to it by the algorithm. This will prove to have advantages for algorithms which will interface with FF and FFD.

4.4.2 On-line Vector Packing

The on-line version of Vector Packing is not well explored in the literature. The only algorithm analysed is FF, which works in the same way as in the off-line case.

4.4.3 Vector Packing and Resource Allocation

In the case of resource allocation in a Hijdra environment, the approximation by casting the problem as a Vector Packing problem falls between the off-line and the on-line version. Although the on-line algorithms could be applied directly, the off-line algorithms need to be adapted to be able to use partially filled bins instead of solely the empty ones which they expect.

The FFD algorithm can be adapted trivially, since it leaves the packing to First Fit, which is an on-line algorithm. Because on-line algorithms pack the items one at a time, there is another advantage over many off-line algorithms. Recall that to be able to cast the resource allocation problem as a Vector Packing problem, it had to be over-dimensioned by using $\forall e \in E : \delta(e) = 0$. For on-line algorithms, this is not necessary, since the size of the actor under consideration can be adjusted to incorporate the $\delta(e)$ for every tile and every channel to an actor which is already packed. Note that this still is an over-dimensioning, since for channels to unmapped actors, $\delta(e)$ is not yet subtracted.

4.4.4 Clustering Strategies

When Vector Packing algorithms are applied to solve the resource allocation problem, there is no control over the amount of bandwidth used by the final solution. To use less bandwidth, the algorithm should try to place connected actors on the same tile. There are three basic moments to do this: before, during, and after the packing phase. These strategies will be named Clustering Before Packing (CBP), Clustering During Packing (CDP) and Clustering After Packing (CAP), respectively. Their interactions with the packing algorithm is shown in Figure 4.1.

In CBP, a set of actors can be replaced by one large actor, which represents this actor set as to force them all to be mapped to the same processing tile. This large actor will be treated by the packing algorithm as any other actor, making CBP transparent to the packing algorithm.

In CDP, the packing algorithm works together with a clustering heuristic to get the right combination of actors on each processing tile, such that the bandwidth needed between the tiles is minimised.

In CAP, the actors can be swapped around after they all have been assigned to a processing tile. This strategy effectively optimises an existing mapping.

Before Packing

Before the actors are packed on tiles, sets of actors can be located which are heavily connected. These are then grouped together such they are forced to be packed on the same tile. This could lead to an increase in the number of

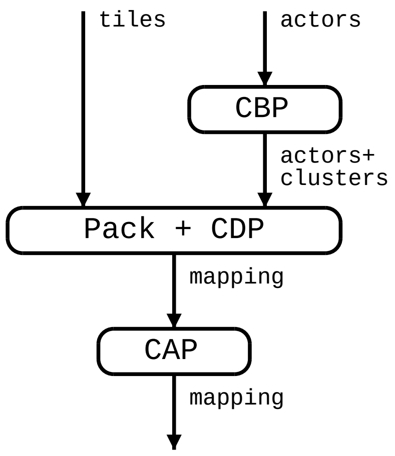

Figure 4.1: The interaction flow between clustering strategies and the packing algorithms.

required tiles due to too aggressive clustering, but it could also lead to a decrease due to the space saved by better use of the δ-function. (heterogeneous systems and tile size).

For homogeneous systems, this can be defined as follows:

Problem 4 (Clustering Before Packing Problem). *Given a graph $G(V, E)$, a partition size F, weight functions w, δ and b, and a natural number k. Does a partition of V into $P = \{P_1, \ldots, P_n\}$ exist, such that for each partition*

$$i = 1, \ldots, n : \sum_{v \in P_i} w(v) - \sum_{\substack{e=(a,b)\in E \\ a,b \in P_i}} \delta(e) \quad \leq \quad F,$$

and the total length of the cut

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{x \in P_i, y \in P_j} b((x, y)) \quad \leq \quad k?$$

The F, w and δ functions are those as defined before. Since every tile is the same, it is omitted as a parameter to these functions.

Theorem 1. *Clustering Before Packing is NP-complete.*

For a proof, see Section B.1. This problem ignores the fact that some tiles can already be partially occupied. There is a danger that the clusters created by algorithms which solve or approximate CBP are rather large (approaching the size of a tile), and cannot be used to augment already partially occupied tiles. If the packing algorithm does want to augment such tiles, it has to split up one or more of the clusters into its individual actors and use those actors for augmentation. In extreme cases, this could render the clustering done useless as breaking up any cluster greatly degrades its quality.

As a special case, CBP is useful if there are individual actors which do not fit on any tile, unless they are packed along with some of their neighbours. For instance, take a channel $e = (a, b)$. If the following holds for an empty tile t:

$$\begin{aligned} w(a) \not\leq F(t) \quad &\vee \quad w(b) \not\leq F(t) \\ w(a) + w(b) - \delta(e) \quad &\leq \quad F(t), \end{aligned}$$

then actors a and b have to be packed together. Such situations can be recognised at design time, and are thus a form of CBP. It is assumed this situation is avoided, or already taken care of, at design time. The RM can thus assume any actor can fit on an empty tile, i.e., $\forall a \in V : w(v) \leq F(t)$ for an empty tile t.

Although CBP does not use information about the situation in the system, like the presence of partially occupied tiles, its use is justified. The packing algorithm is not obliged to use the clusters, since it can still choose to pack the individual actors instead. Using CBP allows the creation of a high-quality clustering at design time, which is useful at run time in several situations. It can be used to map the first job efficiently. Also, it can be used as a fall-back to map the application to a set of empty tiles efficiently, if the on-line algorithm fails to find any better solution. Both situations benefit from a pre-calculated clustering, especially if it is substantially better in terms of bandwidth or tile usage than solutions calculated ad-hoc.

During Packing

When packing the actors onto tiles, the packing algorithm can consider to pack actors which are connected to the ones already mapped to the tile which is being filled. This requires the algorithm focuses to fill bins instead of finding any bin to place each actor.

The First Fit algorithm allows this. When it assigns an actor to a tile, it can be modified to look at that actor's neighbours and try to put them on the same tile. This is in essence changing the order in which actors are considered, while the algorithm is running.

Because this algorithm performs clustering as well as a form of First Fit packing, it will be called FFC.

After Packing

After packing, actors can be swapped between tiles to get connected actors on the same tile. This can be defined as follows:

Problem 5 (Clustering After Packing Problem). *Given a graph $G(V, E)$, a partition size F, weight functions w, δ, and b, and a natural number k. Also given a partitioning of V into partitions $P = \{P_1, \ldots, P_n\}$, each of at most size F, that is*

$$i = 1, \ldots, n : \sum_{v \in P_i} w(v) - \sum_{\substack{e=(a,b) \in E \\ a,b \in P_i}} \delta(e) \quad \leq \quad F.$$

Is it possible to repartition V into n partitions $Q = \{Q_1, \ldots, Q_n\}$, each of at most size F, having a total cut length of

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{x \in P_i, y \in P_j} b((x, y)) \quad \leq \quad k?$$

Theorem 2. *Clustering After Packing is NP-complete.*

For a proof, see Section B.2. The CAP strategy cannot use more tiles than the packing phase before it specifies. This is essential, because requiring more tiles would defeat the purpose of packing first. The algorithm cannot assume there are more tiles available than the packing used.

No algorithm implementing CAP will be tested in this report. If the packing algorithm creates a good mapping using few bins, there is little room to swap actors around. To do this, a local search algorithm could be employed. It was considered more likely that doing CBP and CDP would lead to situations superior to those CAP would obtain, on average.

4.4.5 Clustering Algorithms

A greedy CBP algorithm can be constructed as follows: order all edges non-decreasing by their bandwidth requirement. For every edge, contract it so that the result will still fit on a tile. Contracting an edge $e = (a, b)$ means replacing both actors by a single actor c which represents these two endpoints. The resource requirements of this resulting cluster is

$$w(a) + w(b) \quad - \textstyle\sum_{e=(a,b) \in E} \delta(e) \quad - \displaystyle\sum_{e=(b,a) \in E} \delta(e).$$

Every edge incident to a or b, but not both, will instead be incident in the same way to c.

This greedy clustering algorithm can be controlled by allowing only a certain percentage of the edges to be contracted into a single entity. Given a certain percentage, the algorithm will try to contract up to that percentage of the edges. As a result, a tradeoff can be made between the amount of bandwidth saved directly by this clustering and the average granularity of the resulting clusters. Due to their granularity, smaller clusters are easier to combine on to processing tiles, so not contracting the smaller edges might be preferred.

4.5 Test Setup

The main purpose of the RM is to find a feasible solution for mapping a job, as often as possible. Whether it can do this depends on both the algorithm used and the current state of the system. The number of possible states is enormous. A state is defined not only by the set of jobs running on the system, but also by the job history. For example, if a job X is added to an empty system, it will have a different mapping than if it is added to a system with some jobs already running. If in the latter case all the jobs except X are terminated, again a system with only X running, is left. Since the mapping of jobs does not change during their runtime, job X, even though it is running on an empty system in both cases, can have many different mappings.

Tests will be performed, focusing on two situations. On the one hand, the algorithms will be subjected to mapping jobs to an empty system. This situation is reproducible and easily analysable. But because this does not test the compositionality and fragmentation issues of the solution, a test 'game' is derived to stress the system, forcing it to map jobs under heavy load. This will generate extreme conditions, and the performance of the RM will be measured against them. In all tests, the system will consist of a single router connected to 10 processing tiles.

This game works as follows:

1. Select a set of jobs J which are allowed to run on the system.
2. For each job $j \in J$, calculate the sums $\vec{r_j}$ be the vector of the total amount of CPU and memory requirements of their actors and channels.
3. Let the set of running jobs R be equal to $\emptyset$.
4. Let $\vec{t}$ be the vector containing the total amount of CPU and memory available in the system.
5. Select a job $s \in J$
   R.
6. If $\vec{r_s} + \sum_{j \in R} \vec{r_j} \leq \vec{t}$, the job should fit on the system from a total resource usage viewpoint. If this is not the case, terminate running jobs at random until the inequality holds.

7. Try to map s. If successful, run it, i.e., $R := R \cup \{s\}$.
8. To do another round, go to step 5.

In the first few rounds, the jobs selected will fit and fill up the system. Then, the algorithm will try to add jobs which should fit, where only the totals of the CPU and memory resources are considered. The rate of success, being the number of successful mapping attempts divided by the number of rounds, will be measured.

As is, the game is not a fair one for the RM. The RM cannot be expected to fill up all processing tiles to 100%, even if it would use optimal packing algorithms. Instead, the performance of RM will be plotted against the percentage of tiles it will manage to fill. This will be done by introducing a slack parameter $\sigma \geq 0$, and the inequality in step 6 is changed into

$$(1+\sigma)\left(\vec{r}_s + \sum_{j \in R} \vec{r}_j\right) \quad \leq \quad \vec{t},$$

so the RM is expected to require a factor of δ more resources than the job strictly needs. If such a σ creates a low failure rate, this implies the system designer should over-dimension the system to have $\sigma\vec{t}$ resources, if the designer wants the system to be capable of running a set of jobs with a total size of $\vec{t}$ concurrently.

The decision of which jobs to allow in the game is subject to a set of observations. First, if a mix of small and big jobs is used, the game is allowed to stop a big job to make room for a small one. The RM will find a location for the small job relatively easy due to the abundance of free resources. Any subsequent jobs, if small, will also be easily mapped. Secondly, if a set of solely large jobs is used, a granularity problem arises. For instance, if all jobs require 4 tiles, and the system contains 10, the system allows a lot of slack in itself. It will not expect the RM to be able to map three jobs at a given time, and has plenty of resources to map two of them. Such a situation will hardly test the quality of the RM. Thirdly, for a set of larger jobs, the difference between the largest and smallest job in the set increases, which leads to the first problem. For these reasons, the game will be tested using a set of jobs consisting of only 10 actors.

4.6 Empirical Results

The Vector Packing problem is a subset of the resource allocation problem. As such, the approximation algorithms for Vector Packing can still attain the same worst-case performance (or even worse) when used for resource allocation. These worst-case bounds are quite high: in the case of 6 dimensions, First Fit can require up to 6.7 times more tiles than the optimal solution. Fortunately, in practice this will not be the case.

The next sections will define the test instances, and evaluate the algorithms that were described in this chapter.

4.6.1 Job Instances

Because of a lack of real application graphs, random ones need to be constructed with properties which are likely to be found in real applications. These general properties were, for a graph $G(V, E)$:

- An edge density of about 1, i.e., $|V| = |E|$.
- A size of $|V| \in \{10, 20, \ldots, 100\}$.

The single-router networks do not have as many tiles as the multi-router networks. The tests in the single-router case will therefore be restricted to graphs of 40 actors. The larger graphs need larger systems, and will be tested once multiple routers, and thus more tiles, are allowed.

For the resource requirements of both actors and channels, several classes were defined:

Class I Heavy actors, heavy edges.

Class II Heavy actors, light edges.

Class III Light actors, heavy edges.

Class IV Light actors, light edges.

The resource requirements of both the actors and the edges are chosen from a uniform random distribution. A heavy actor is an actor which uses 4–14% of the CPU cycles and memory of a tile. A heavy edge uses 2–7% of the available memory and bandwidth of a tile, as well as 2–7% of the bandwidth available in the CA. The light actors and edges use half the amount of resources as the heavy versions. Each processing tile will be able to keep track of 30 connections over the network.

For every class and instance size, 100 instances were randomly created. No attempt was made to create DAGs. None of the algorithms is exploiting this property, and since these applications are artificial, there is no concern about their theoretical throughput. It is assumed that the results measured on these graphs can be considered the same as for DAGs.

4.6.2 Packing Algorithms

First, the performance of various packing algorithms was tested without applying any clustering. The problem is over-dimensioned by assuming $\forall e \in E : \delta(e) = 0$. Informally, this means there is no gain in resource usage if a

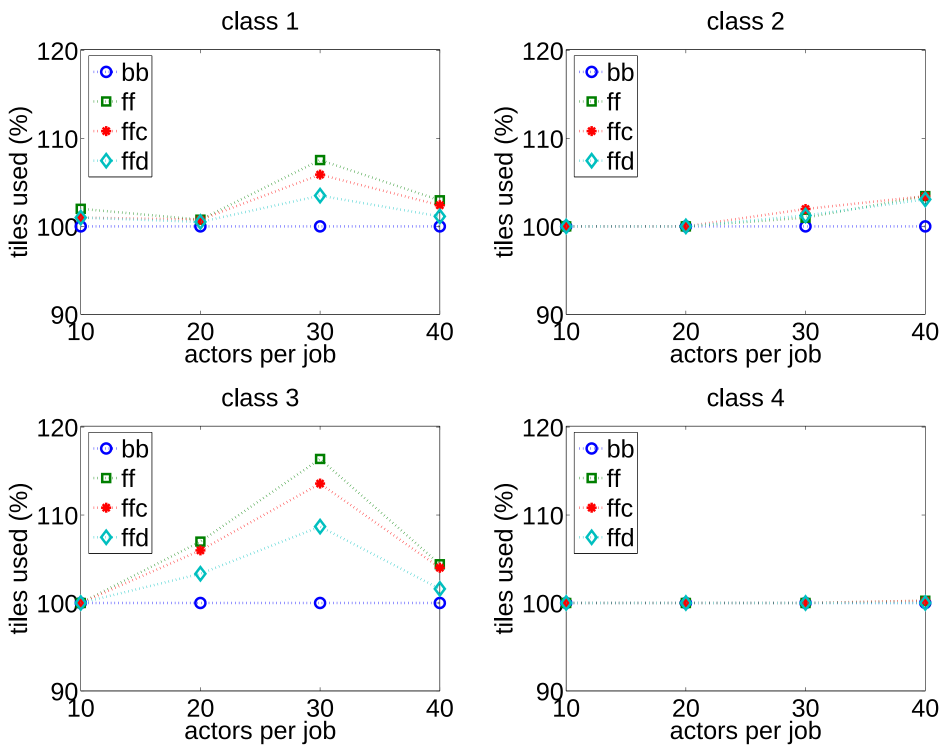

Figure 4.2: The percentage of tiles used to host jobs of increasing size. The optimal solution represents 100%.

channel is mapped internally. This allows an exact solution to be found by a branch-and-bound (BB) algorithm. This solution is then compared to the performance of FF, FFD, FFC, BF and BFD for all classes. The result of this can be seen in Figure 4.2. The Best Fit algorithms were found to have a performance nearly equal to their First Fit counterparts. Since the First Fit algorithms are less computationally expensive, the Best Fit algorithms were omitted from the graph.

In this figure, the horizontal axis represents the size of the instance, and the vertical axis represents the number of tiles which were needed to host the instance, in percentages relative to the optimal value.

These graphs appear to imply that the actual average performance of the approximation algorithms tested is far below the worst case given by theory. All algorithms tested need, on average, less than 20% more tiles than optimal while FFD (and BFD) require less than 10% more. This is far better than the 533% increase which can occur for FFD and BFD, according to theory. In all cases, if the approximation algorithms could not find the optimal solution, they only needed at most one tile more.

The reason that such extremely bad behaviour is not encountered in empirical tests, is twofold. First, it has been shown that (at least in the single-dimensional case), on average, algorithms like FF and FFD perform far better than their theoretical worst case [5]. Secondly, a possible increase of 533% stems from the fact that 6-dimensional vectors are being packed. Theoretically, each new dimension can add additional problems for the packing algorithm and thus increase its worst-case performance. In practice, not every dimension will turn out to be of equal influence. For instance, if all processes use relatively more CPU cycles than memory, the packing algorithm can ignore the memory dimension as it will always fit if the CPU cycles do. This results in one less dimension to pack, but the worst-case analysis ignores such facts.

This result motivates the use of these very simple algorithms, removing the need to test more complicated algorithms like CK and FVL. These algorithms are not likely to provide a gain in performance which is worth their complexity.

Internally mapped channels

The next step is to see how much performance can be gained by allowing $\forall e \in E : \delta(e) \geq 0$. This allows the packing algorithms to save on resources (network interface slots and bandwidth) if they map a channel internally. The resulting problem is no longer bin packing, making it impossible to use BB to solve it. Instead, the results for BB from Figure 4.2 are copied as a reference point. The approximation algorithms do allow this change to be incorporated, as explained in Section 4.4.3.

The result of this is shown in Figure 4.3. The gain in tile usage is especially visible in class 3, in which the algorithms now perform within 5% of the old optimum, compared to around 18% in Figure 4.2. The actual optimum in this graph lies somewhere below the BB-line, but is complex to compute. The algorithm for BB relied on lower bound computations provided by literature. These lower bounds no longer hold in the case of $\delta(e) > 0$. Such a test has therefore been omitted in this work.

The approximation algorithms show an improvement in their performance. The FFC algorithm can now take advantage of a decrease in resources if it manages to map more channels internally. Its performance is now nearly equal to that of FFD, which had an advantage over FF due to sorting the items in non-increasing order before packing them.

4.6.3 Clustering

The effect of clustering is measured by using the set of largest graphs (40 actors) for each class and applying the CBP greedy clustering algorithm in varying degrees. The CBP algorithm attempts to contract 0% to 60% of the channels, in 1% increments. If it is not possible to cluster the requested

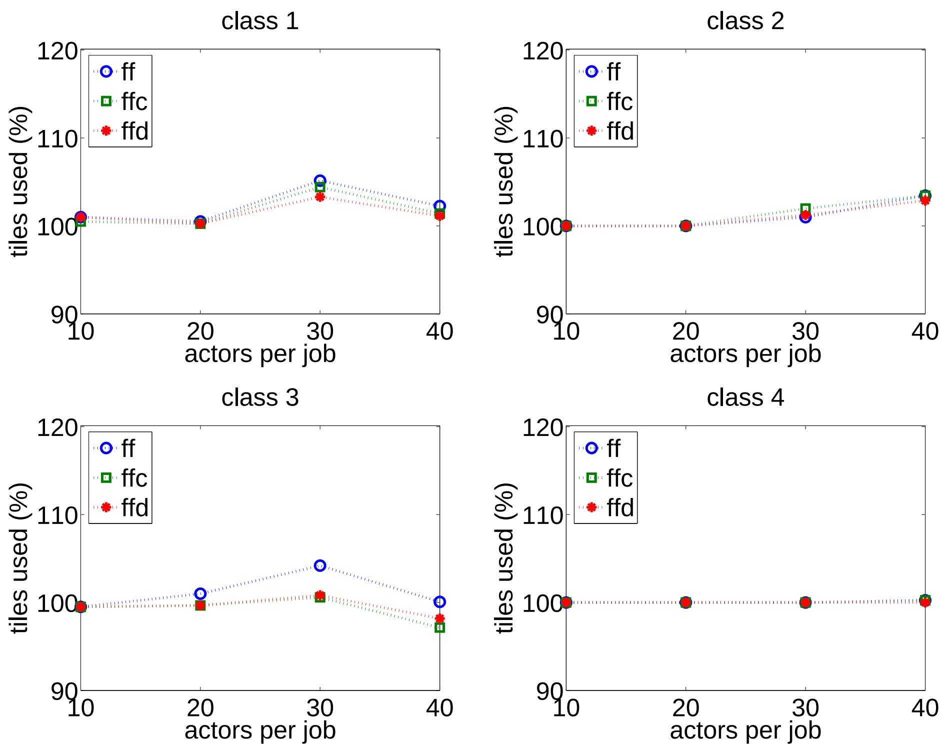

Figure 4.3: The percentage of tiles needed, if it can save resources by mapping a channel internally. The results are shown against the BB solution from Figure 4.2.

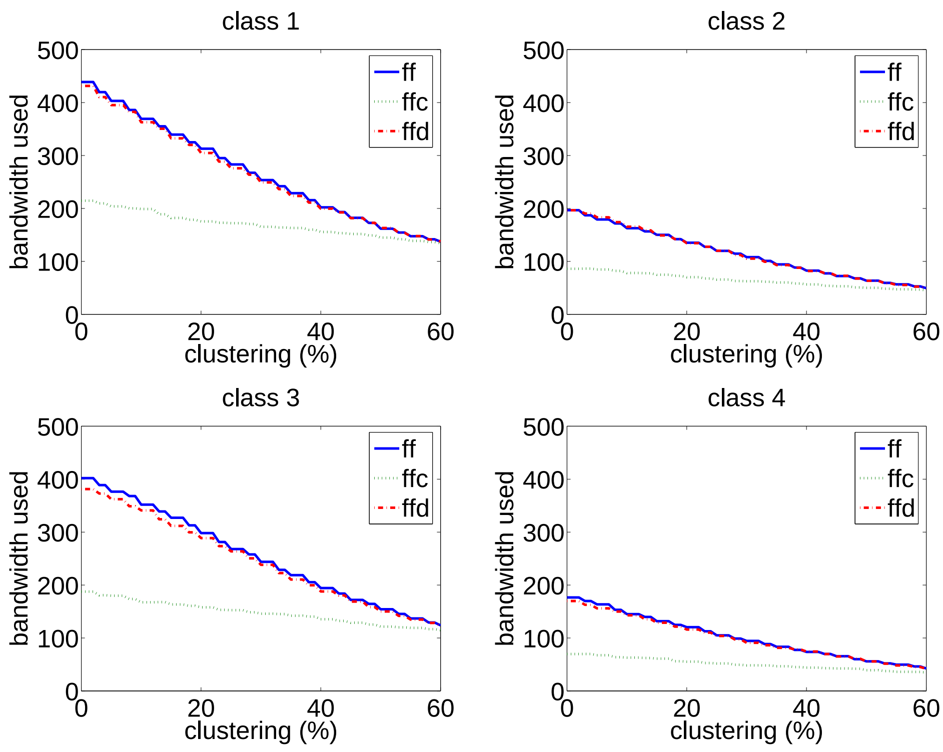

Figure 4.4: The bandwidth used, as a function of the clustering percentage.

percentage of channels, as many channels as possible are contracted. Trying to contract more than 60% of the channels succeeded in none of the cases.

The packing algorithms FF, FFD and FFC were all tested in this context. Of these algorithms, FFC includes a CDP clustering strategy which augments the CBP greedy clustering.

The bandwidth required after clustering and applying any of the packing algorithms, is shown in Figure 4.4. The difference of average bandwidth usage between FF and FFD is next to none. This is no surprise considering these algorithms totally ignore bandwidth usage. The advantage of using a clustering algorithm for these cases however is evident. It allows an average drop of bandwidth usage between 60% and 80%, depending on the job class.

In the same graph, results for the FFC algorithm are plotted. This algorithm perform packing and clustering simultaneously. It comes as no surprise that when no greedy clustering is employed, this algorithm outperforms both FF and FFD in terms of bandwidth usage. When greedy clustering is added, it augments the clustering performed by FFC. The gain in bandwidth usage for FFC lies between 17% and 44%. At the highest clustering percentage, the performance of FFC and FF/FFD are nearly indistinguishable.

The reason for this is twofold. First, the average size of the clusters increases as the percentage of greedy clustering increases. With larger clusters, it will be more difficult to combine neighbouring clusters on the same tile. In fact, at the highest percentage this is impossible, since the clustering algorithm would have merged them into a single, bigger cluster before FFC would get that chance.

Secondly, the greedy algorithm can outperform FFC in terms of bandwidth usage, since it uses global knowledge about the task graph at hand. On the other hand, FFC only uses local knowledge, as it tries to add the neighbours of the actors already mapped to a tile. In some cases, it is beneficiary to not add such a neighbour, if that neighbour communicates heavily with a third actor which would not fit on the tile with the first two actors combined.

While heavy clustering reduces bandwidth usage, there is a price to be payed in the number of tiles which the packing algorithm requires. In Figure 4.5, the number of tiles needed to map the job is plotted against the clustering percentage. It shows a less than 5% increase in the number of used tiles if full clustering is used. This allows for an easy tradeoff in the optimality Formula 4.1, as the tile usage is nearly constant.

4.6.4 The Game

The tests done so far assumed the system is empty when a job is to be mapped, and tried to reduce the resource usage for that case. To test the performance of the algorithms on a non-empty system, the stress test game as described in Section 4.5 is run. The game is run for 10000 rounds.

The tests are run using FFC to pack the actors, since that was the most promising of algorithms, which does not depend on greedy clustering, in the previous tests. Different amounts of clustering are tested against different amounts of slack, resulting in Figure 4.6.

This graph shows several remarkable results. First, if no clustering is used, the system only needs 6–10% slack to eliminate virtually all of the failures. The algorithms thus prove to be amazingly efficient, regardless of any difficult fragmentation issues that could theoretically occur. Apparently, either fragmentation does not pose a problem, or the hard cases are not encountered in practice.

Secondly, CBP does not seem to work as well as suggested by previous tests. It worked in the single job tests, the job was mapped to an empty system. This implies the clusters created could always be placed. In this game test, the clusters created by CBP are too big to put on processing tiles which already clusters actors from other jobs. If the system is at a high load, this could mean enough resources are available in the system, but they are fragmented. A newly arriving job may fit if the RM looks at the total amount of resources available, but the clusters created are too large to fit anywhere. The RM will have to be able to map jobs even if most of the

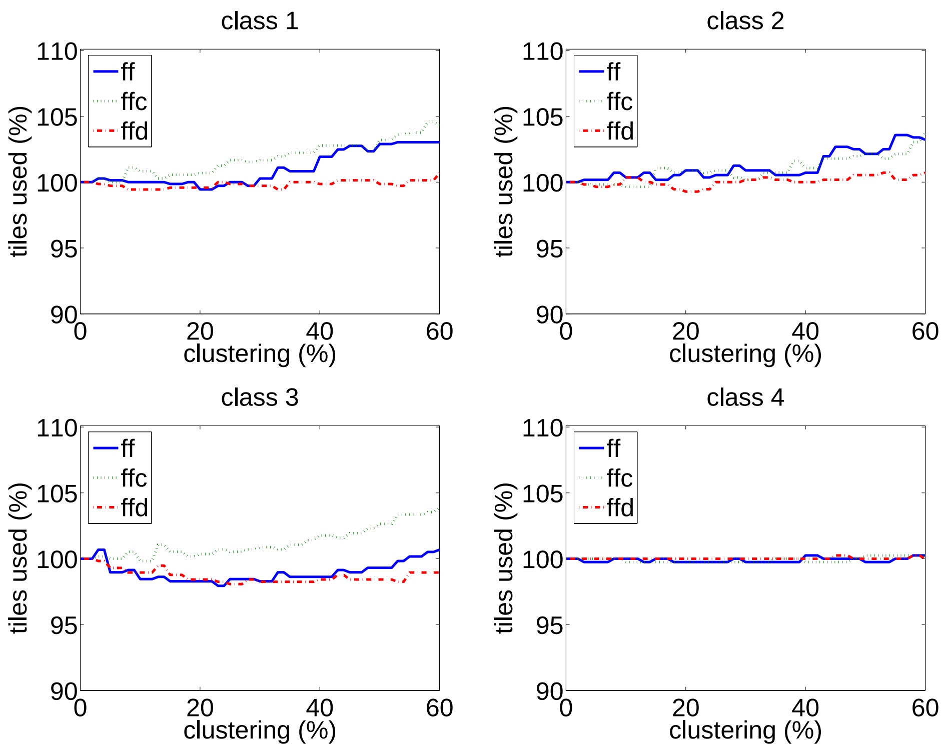

Figure 4.5: The percentage of tiles used, as a function of the clustering percentage. For each algorithm, the percentage of tiles it needs at 0% clustering is given the value of 100%.

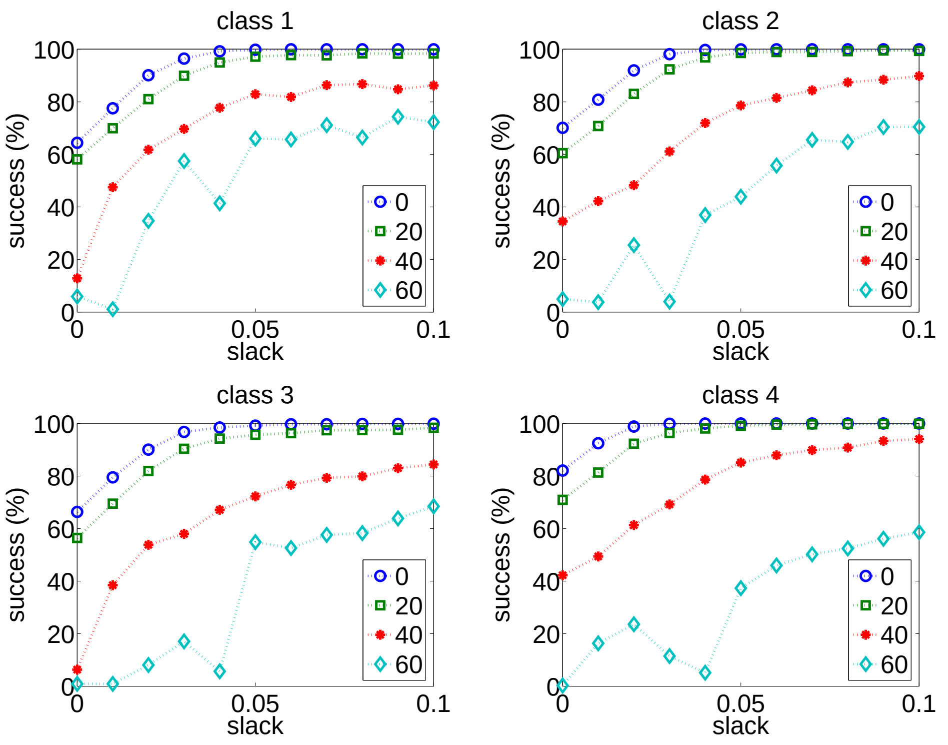

Figure 4.6: The percentage of successful mappings, as a function of the allowed slack. Each line represents a clustering percentage.

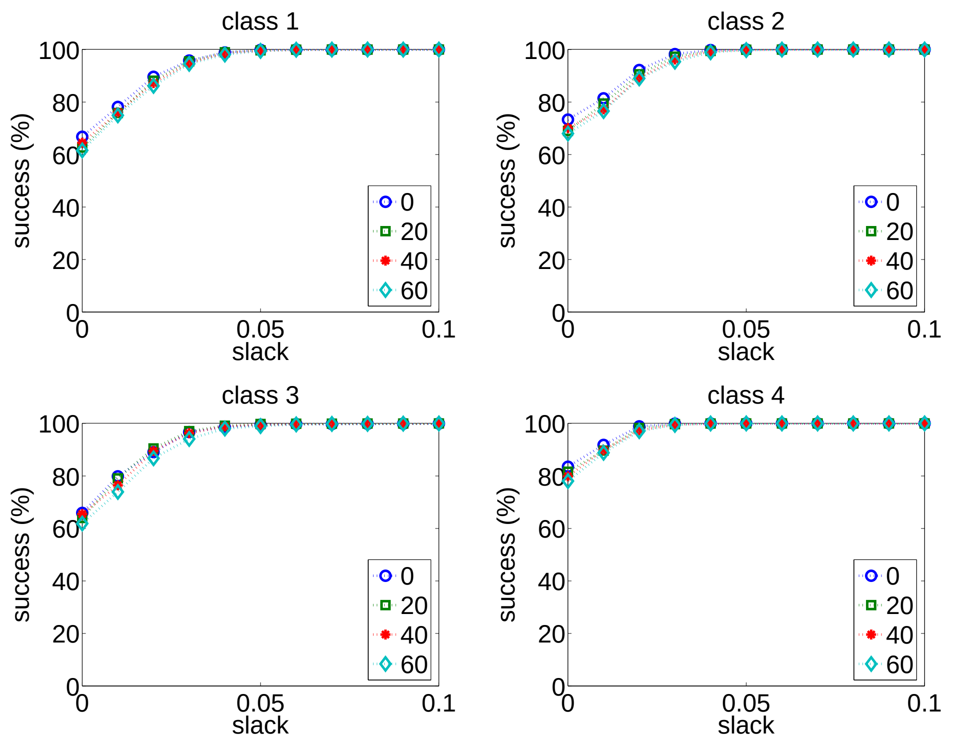

Figure 4.7: The percentage of successful mappings, as a function of the allowed slack. Each line represents a clustering percentage.

available tiles are partially occupied.

Thirdly, the graph occasionally spikes downwards, which would not be expected if more slack is allowed. A case analysis showed this is due to the game getting stuck into a local minimum, in which the jobs running are mapped badly. According to the amount of free resources, the RM should in these cases be able to add another job, but it fails doing so for many jobs in a row. This accounts for a streak of failures, of which the impact is visible in the graphs. This effect occurs more often at high clustering percentages, as they are less suited to deal with fragmented mappings.

This bad behaviour can be largely compromised by using a fall back: if the RM cannot map the clustered version of the job, it should try to map the unclustered version. The resulting performance can be seen in Figure 4.7. With this modification, clustering can now compete with the unclustered packing, and even improve on it.

The tests in the previous sections implied that the strength of clustering lies not in optimising tile usage, but in optimising bandwidth. If the system tested had ample bandwidth available, the mappings can succeed even without optimising for bandwidth. Would the jobs require more bandwidth, this

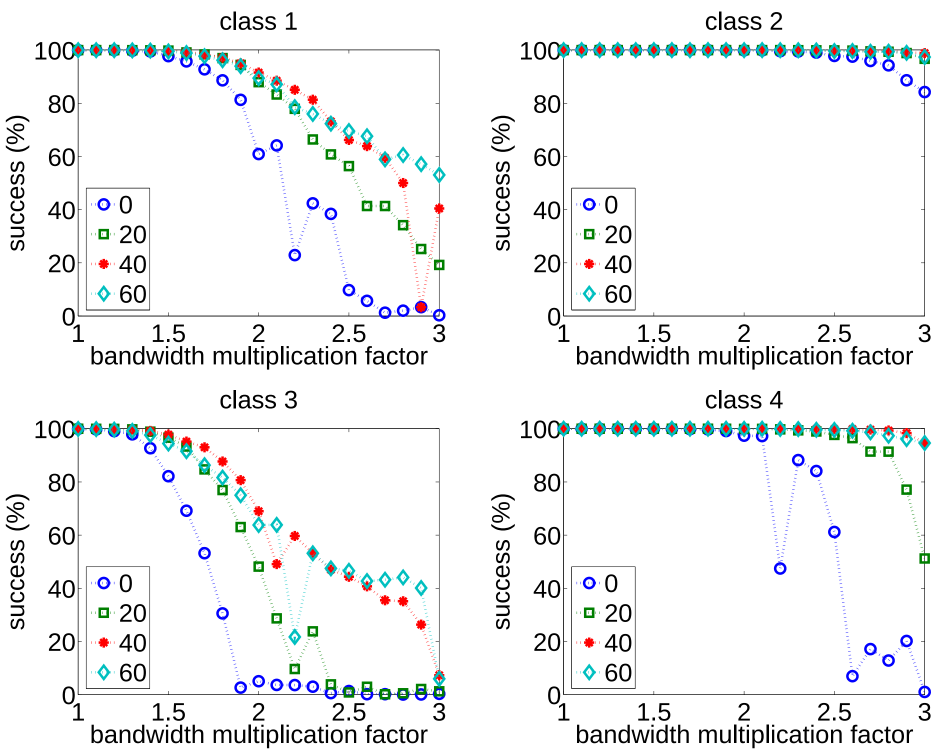

Figure 4.8: The percentage of successful mappings, as a function of increased bandwidth requirements. Each line represents a clustering percentage.

could become critical, and clustering might start to pay off. If the bandwidth requirements of all channels are scaled up, and allow for 10% slack, the performance results in Figure 4.8.

The increase of the bandwidth required per channel changes the picture entirely. At some point, depending on the class of jobs, greedy clustering cannot be omitted if the mapping is to succeed. In class IV, eventually the RM failed to map almost all jobs if not some form of clustering is employed. With enough clustering, it was capable of mapping almost all of them. In classes I and III, even clustering is not enough to map the instances, but it should be noted that no way has been provided to prove the feasibility for such jobs.

4.6.5 Conclusions

The family of First Fit packing algorithms have shown to form a simple but effective basis to solve the resource allocation problem in single-router networks. Unmodified, First Fit sees the bandwidth usage as of the mapping as a constraint to satisfy per processing tile, but cannot optimise for it. To optimise for bandwidth usage, First Fit was modified such that when it maps an actor to a processing tile, it tries to map its neighbours there as well. This variant was named FFC.

Also, it is possible to calculate, at design time, a partitioning of each job graph, such that each partition fits on a processing tile, and the bandwidth used between the partitions is minimised. Mapping these partitions instead of the actors proved to decrease the bandwidth usage of the mapping even more. However, empirical tests show that the RM has to be able to fall back to packing the original actors, since the partitions do not always fit into a partially occupied system.

If bandwidth is a scarce resource compared to the requirements of the job which is requested to be mapped, this combination of employing a pre-calculated solution and falling back to FFC proved by far to be the best solution. If bandwidth is available in abundance, this pre-calculated solution is not necessary but can provide a decrease in power consumption. The test showed that it does not hurt to try.

Finally, in the configurations tested, this combined solution has shown to be capable of using at least 95% of the computational resources. This effectivity decreased in situations in which bandwidth was a scarce resource, but the existence of a pre-calculated solution still helped the RM to achieve a high success rate in many cases.

Chapter 5

Multiple Router Networks

A single-router network as described and analysed in the previous chapter covers only part of the full resource allocation problem as it exists in Hijdra. Instead of a single router or a bus, a network of routers will exist between the processing tiles. Any channel which is mapped externally on such a system requires a dedicated route through this network between both of its endpoints.

This means that even in a homogeneous system, the tiles are not indistinguishable anymore. Algorithms have to be devised in order to determine in which tile an actor is going to be placed. While the algorithms presented for single-router networks could serve this purpose, they do not specify which tile should be chosen if a new one is used for the first time: there was not any choice to be made.

This chapter models the network and analysis algorithms for discovering routes between two tiles. Additionally, the packing algorithms which were promising in the single-router case will be extended to include topology knowledge in order to improve their decisions. The effectiveness of all this shall be empirically tested for several topologies.

5.1 Network Model

The network in Hijdra systems is a directed graph of routers, tiles, and the links connecting them. The topologies that can be formed are limited by the fact that every tile can be connected to only one router, and every router can only have a limited number of links.

Every router contains a full interconnection network making it able to send data from any input link to any output link simultaneously. This interconnection network comes at a physical cost, scaling the router quadratically on the number of links. A limit of 8 links per router is considered to be a realistic case.

5.1.1 Routing

All routers and tiles send and receive data synchronously as if operating under a global clock. If data arrives on an input at tick t, it will be sent through the router to the output link and arrive at the other end of that link at tick $t + 1$.

In the tiles, the Network Interface (NI) serves as the intermediate between the network and the memory. A NI contains a set of FIFO buffers, which contain the data for the different SDF channels allocated to send data to other NIs. A TDMA wheel is present as well, which divides the available bandwidth at the ticks of the global clock. For each FIFO buffer, slots in this wheel can be reserved to reserve bandwidth. Because every NI has a TDMA time wheel of equal size, the whole network can be considered to be subject to the rotation of these wheels.

The route through which the packets of a FIFO buffer flow through the network should be allocated, and is fixed for the lifetime of the SDF channel it supports. Such a route through the network consists of a simple path defined by the routers used and the slots in the wheel used at the source NI. To avoid collisions between routes, they should be allocated not only per hop but also per slot in each router's time wheel.

The network graph will therefore be expanded to include these time slots as well. Each router vertex is replaced by T vertices, where T is the number of slots in the time wheel. Each edge is replaced by T edges. If the edge is between two routers, it connects every slot vertex t of the source to slot vertex $(t+1) \mod T$ of the sink. If the edge is between a router and a tile, each edge connects slot vertex t of the router to the single vertex representing the tile. The Figures 5.1 and 5.2 show such a transition for a case with two routers, two tiles and $T = 4$.

5.1.2 Packets

The packets travelling through the network consist of a header and a payload. The header can take up a significant portion of the bandwidth: situations in which the header is one word, and the payload is two words long are realistic. To save bandwidth, a NI can allocate adjacent slots in its time wheel, which are configured to follow the same route. This implies adjacent slots will be used from the source NI to the sink NI. In such adjacent slots, it is possible to send a header in the first, and let the payload span the rest of all the bandwidth of all the adjacent slots allocated.

5.2 Route Discovery

When the source and sink of an SDF channel are mapped to different tiles, the channel requires a route through the network to be allocated between

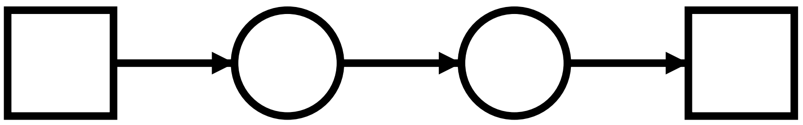

Figure 5.1: A network graph of two tiles and two routers, without any slots modelled.

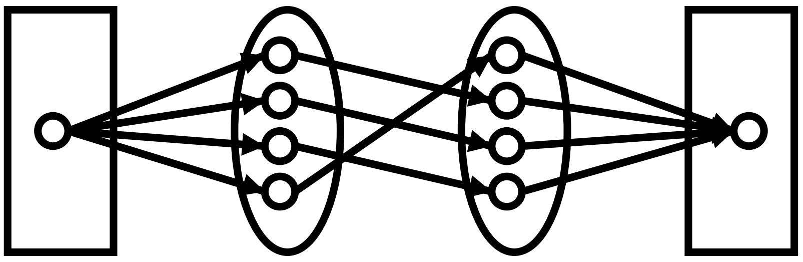

Figure 5.2: A network graph of two tiles and two routers, with 4 slots modelled per node. Each router is replaced by 4 nodes, each representing a time slot and is either connected to the next time slot in the next router, or to the node representing the tile.

these two tiles. In the network graph model, this corresponds to finding a set of paths between the two NIs of those tiles, which together provide enough bandwidth to support the channel.

The problem of finding paths through digraphs is well known in graph theory. In particular, finding the shortest path between two vertices is a useful case:

Problem 6 (Shortest Path Problem). *Given a weighted digraph $G(V, E)$ and two vertices $s, t \in V$. Find a path from s to t of minimum weight.*

Since all the edges in the network model provide the same bandwidth, the weights of the edges can be considered 1. This allows Dijkstra's algorithm to be used to find the shortest path between any two NIs in $O(|V|^2)$.

For the whole job, generally more than one route will be allocated through the network. These routes are not allowed to collide, so between the several pairs of source and sink vertices, edge-disjoint paths need to be discovered:

Problem 7 (Directed Edge-Disjoint Paths Problem). *Given a weighted digraph $G(V, E)$ and a digraph $H(V, F)$ using the same vertices. Find a set of edge-disjoint paths in G, containing exactly one path from s to t for every pair $(s, t) \in F$.*

5.2.1 Complexity

The Directed Edge-Disjoint Paths Problem is NP-complete, even in many restricted cases [11]. In the resource allocation problem, more freedom is allowed: the RM is free to choose on which tiles the actors will be mapped, and thus free to choose the location of the sources and sinks of every channel. This leads to a different problem, specifically:

Problem 8 (Job-to-Network Mapping). *Given a job digraph $G(V, E)$, and a network topology $T = (P \cup R, N)$ of processing tiles P, routers R and a network N. Also given a weight $b(e) > 0$ for each edge in E and a capacity $c(n) > 0$ for each link in N. The topology T is restricted by the fact that each vertex in P is only connected to exactly one router in R (assuming $|P| > 1$). Does there exist an injective mapping of actors V to tiles P such that there exist edge-disjoint routes through the network (through R and N nodes) to accommodate the edges between the actors in the mapping.*

Theorem 3. *Job-to-Network Mapping is NP-complete.*

For a proof, see Section B.3. This proof also shows the NP-completeness still holds even if each actor has to be mapped to a different tile, and both the job and network graph are trees.

The JNM problem will be tackled by using a layered approach. The individual tiles will be filled by the clustering and packing algorithms as described in Chapter 4. The RM will decide about the actual placement of the contents of each of these tiles. Since many tiles are similar, it can move the content of one tile to another one. Once the location of the contents of one or more of the tiles is known, routes between them can be discovered. This last step is equal to the Directed Edge-Disjoint Paths Problem, and due to its complexity, approximations have to be used.

5.2.2 Approximation Algorithms

Instead of trying to solve the Directed Edge-Disjoint Paths Problem directly by considering all paths at once, it is approximated by allocating routes one by one. Each time a Shortest Path algorithm is used to find paths for each channel, and the shortest of these shortest paths is chosen and allocated. This algorithm is called *Greedy Path*. It is capable of finding at least a fraction of $\Omega(1/\sqrt{|E_0|})$ of the routes possible, where $|E_0|$ is the number of channels used by an optimal solution [10].

5.3 Virtual Tile Placement

The clustering and packing algorithms create a partitioning of the set of actors, such that each partition fits on a processing tile. If the network consists of a single router, it does not matter which partition is assigned to which tile, since they can be considered equal. If the network consists of more routers, this is no longer true, because routes through the network have to be found as required by the mapping. In that case, the RM has to decide which partition is to be mapped to which processing tile.

The partitions will be called *virtual tiles*, as they are considered to be processing tiles by the packing algorithm, but do not have a fixed location in the system. Different virtual tile placement approaches allow for a locating a suitable placement of these virtual tiles at three moments in time:

- *On-line*: each time a new virtual tile is required, the RM selects a processing tile in the system. Combined with the packing and clustering algorithms, this results in a flow as shown in Figure 5.3

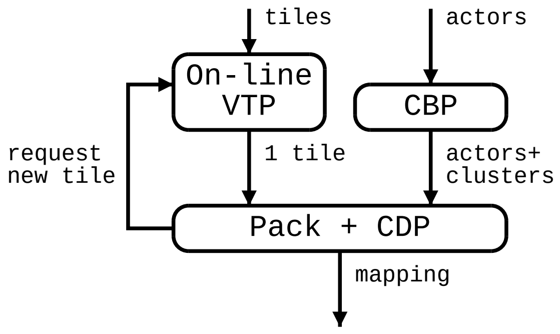

Figure 5.3: The interaction flow between clustering strategies, the packing algorithms and on-line virtual tile placement.

- *Semi on-line*: the packing algorithm fills one virtual tile at a time. Each time a virtual tile is filled up, the RM can find the best location for it. This results in the flow shown in Figure 5.4.
- *Off-line*: when all required virtual tiles are filled, the RM can assign a processing tile to each one. This is shown in Figure 5.5.

The semi on-line and off-line variants assume that all available processing tiles in the system are either totally full or totally empty. This stands in contrast to the required run time requirement of being able to map a job to a non-empty system. However, both variants allow an adaptation in which partially filled tiles are filled up using the on-line variant and fixed in position. The rest of the actors is treated with the semi on-line or the off-line approach. These adaptations will be discussed in Section 5.3.4, but first, algorithms using these approaches will be presented.

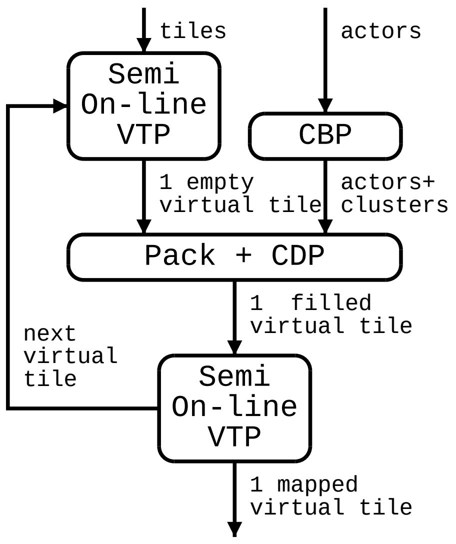

Figure 5.4: The interaction flow between clustering strategies, the packing algorithms and semi on-line virtual tile placement.

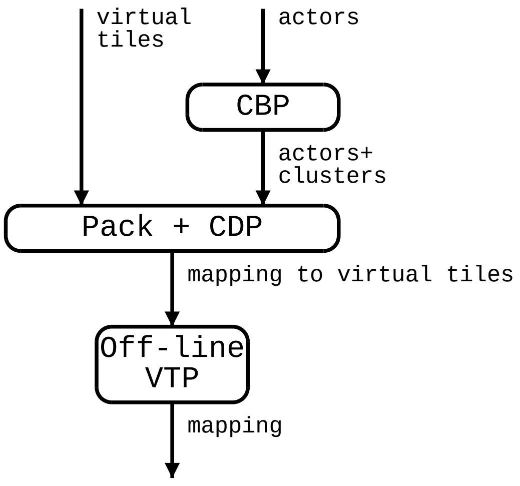

Figure 5.5: The interaction flow between clustering strategies, the packing algorithms and off-line virtual tile placement.

5.3.1 On-line

The on-line packing algorithms work by choosing a processing tile for each actor at a time. These algorithms request new processing tiles once an actor does not fit in the already used tiles. Choosing the location of a new processing tile when the packing algorithm request one is a both simple and natural extension.

When the RM has to choose a new processing tile, it has little information to decide its location. It knows one of the actors that is going to end up on it, but nothing more. Of this single actor, the locations of its neighbours is only known for those that are already mapped.

The heuristic chosen is to choose the processing tile closest to the ones already used. If the mapping started on an empty system, the order in which processing tiles are chosen is thus always the same. This order can be calculated beforehand.

5.3.2 Semi On-line

The FF-based packing algorithms allow a variant in which, repeatedly, a virtual tile is filled as much as possible, and then considered no further. Once the packing algorithm completes a set of actors L to be placed on a single virtual tile, the RM can find a suitable processing tile to host it.

Using this way of packing, the RM knows all the actors that are going to reside on a virtual tile before it has to choose its final location. This allows the RM to choose a processing tile which allows efficient routes between the actors L and all actors N which are already mapped and connected to an actor in L.

The heuristic designed for this case calculates a score for each empty processing tile, equal to the sum of distances from it to the tiles which contain an actor from N. Each of these distances has a weight equal to the bandwidth that is required between both processing tiles. The processing tile with the lowest score is the one on which L shall be mapped.

5.3.3 Off-line

The semi on-line approach can be extended into letting the packing algorithm create a set of virtual tiles, without fixing any of their locations. The RM should then assign a processing tile to each of these virtual tiles, such that the required routes between them can be found.

This problem is NP-complete in general, but the network topologies which will be considered admit certain properties which can be exploited. Typically, each router which is connected to a processing tile is connected to more than one. For the routes that do have to travel over the network, it is favourable to map both endpoints to processing tiles which are connected to the same router.

So again a partitioning is called for. In this case, each partition will be mapped to the processing tiles connected to the same router. The partitions can therefore contain a number of virtual tiles equal to the number of unoccupied processing tiles connected to each router. The bandwidth used between the routers should be kept to a minimum.

In the literature, this problem falls under a class of problems known as the Multi-Way Partitioning Problem. This problem asks for a partitioning of a graph into k pieces, optimising some cost criterion based on the weights of the edges in the cut and/or the sizes of the partitions. A common way to solve this class of problems is to use recursively use a Minimum Bisection algorithm, which can cut the graph into 2 pieces, optimising the weights of the edges in the cut. A well-known heuristic for Minimum Bisection is an algorithm designed by Kernighan and Lin [9]. This algorithm improves on a given bisection by constructs a list S, which contains pairs of vertices, one from each partition, to be swapped. Once this list is constructed, it will swap all the vertices in the list. This is done multiple times, until no more improvements can be found. It works as follows:

1. Create any bisection of V into P_1 and P_2 with $|P_1| = |P_2|$.
2. Set all vertices to 'unlocked'.
3. Let S be equal to $\emptyset$, which is a list. Let G be a list, equal to $\emptyset$.
4. For every pair of unlocked vertices $a \in P_1, b \in P_2$, calculate D_{ab}, which is the reduction of the weights of the edges in the cut, were a and b to be swapped.
5. Find $a \in P_1, b \in P_2$, for which D_{ab} is maximal. Append (a, b) to S. Append D_{ab} to $G_{|S|}$. Swap and lock a and b.
6. Repeat the previous two steps until all vertices are locked.
7. Find $n := \arg\max_{0 \leq n < |S|} \sum_{i=0}^{n-1} G_i$. If $\sum_{i=0}^{n-1} G_i \leq 0$, the algorithm terminates.
8. Swap back all pairs in S after index n.
9. Goto step 2.

The power of this algorithm lies in the fact that it is capable of going around certain local minima; it allows the cut to grow, if it knows it will increase more in the future moves. This happens when, for some $0 \leq i < n$, $G_i < 0$, but still $\sum_{i=0}^{n} G_i > 0$.

The original KL algorithm as described here can only split a graph into two pieces of equal size. It can be adapted to be able to create partitions of unequal size. To do this, the initial bisection should create partitions of the proper size, and the modified KL algorithm should consider all possible swaps in each step. In every step, a pair of nodes is locked, so a swap list of size $|S| = \min(|P_1|, |P_2|)$ will be considered.

This modified KL algorithm inspires an heuristic for grouping the result of the packing algorithm into groups which have sizes equal to the number of empty processing tiles connected to a router. The heuristic considers a router and the set of virtual tiles still to map. It uses the modified KL algorithm to cut off a number of virtual tiles equal to the number of tiles connected to the router. These virtual tiles are placed, and the heuristic considers the next router and the remaining virtual tiles.

According to literature, several techniques exist to make the KL algorithm run faster [6], or perform better [2]. These techniques have not been incorporated in this work.

Note that this approach only shifts the problem to the next level. Up to here, algorithms have been devised to decide which actors are placed in which tiles, and which tiles are placed along the same router. This leaves still the choice of which routers should be picked, and which group of tiles should be placed along each. To provide an answer to this question, more characteristics of the topology are required. It will not come as a surprise that in general, this left-over problem is still NP-complete.

5.3.4 Run-time Allocation

The semi on-line and off-line tile ordering approaches both assume that the packing algorithm will need fresh tiles. If there are already jobs running on the system, the packing algorithm can try to fill up partially occupied tiles. The semi on-line and off-line approaches have to be capable of dealing with such situations.

Adapting the semi on-line approach

The semi on-line approach has to choose the final location of a virtual tile once it is filled by the packing algorithm. If the packing algorithm wants to put actors on a processing tile which already contains actors from other jobs, it can do so. In that case, the contents of the tile cannot be moved to another location. The semi-online algorithm will simply appoint the current location as the only one possible.

Adapting the off-line approach

The off-line approach takes the total collection of all virtual tiles created by the packing algorithm, and distributes them over the available processing tiles, by partitioning them into groups to be placed around each router. If some of the virtual tiles already have their location fixed due to containing actors of previous jobs, this has to be incorporated into the KL partitioning algorithm.

To do this, it first ensures these fixed virtual tiles are in the right initial partitions P_1 and P_2. It then locks these fixed tiles, such that the KL algorithm is not allowed to use them in exchanges. This way, these fixed tiles are not moved by KL, yet their externally mapped channels are taken into account in the optimisation.

5.4 Recovery

If the algorithm cannot find a feasible solution straight away, it should try to explore different decisions. All algorithms presented so far are deterministic. Those algorithms which do not sort their input, like FF and FFC, can obtain different results for different permutations of the input. If the RM fails to find a feasible solution, it can therefore shuffle the ordering of actors in the job and retry.

5.5 Topologies

The Hijdra Architectural Template does not prescribe any network topology, but cannot support every type. The network listens to a global clock in which each router copies a fixed amount of data per time slice. This forces every link in the network to have the same width. Due to physical limitations, as described in Section 2.1.2, a router can be connected to at most 10 inputs and 10 outputs. The size of such a router grows quadratically. For this reason, the topologies tested will be connected to at most 8 inputs and 8 outputs. This allows the placement of more routers, which avoids the tight bottlenecks between the routers if only 2 or 3 routers were to be used. After all, the system has to support more than 10 processing tiles, so it cannot permit to spend a large number of links between the routers if only 2 or 3 routers are present in the system.

The resource allocation algorithms will be tested on a ring of 4 routers. Each router can have as many as 8 connections in both directions, two of which are used to connect to other routers in the ring. This allows for a system of 24 processing tiles. The exact cost of producing such a topology is heavily implementation dependent, so no cost function for the network will be assumed.

5.6 Empirical Results

The performance of the algorithms presented will be tested in the same way as in the single-router case. The first set of tests will map large jobs to an empty system. Since the multi-router systems are larger than the single-router ones, it is possible to map larger jobs. For that reason, the empty system will be subjected to jobs consisting of 100 actors each. For 100 of such jobs, the RM will map and unmap each one in turn, such that each job will be mapped to an empty system. The averages over these cases will be presented.

The second set of tests will run the stress test game, as described in Section 4.5. Again, a set of small jobs (of 10 actors each) will be used in this game.

5.6.1 Topologies

The difference between the use of different topologies was tested first. Table 5.1 shows the different configurations that were considered. A 2-by-2 mesh topology with 8 links per router and 24 tiles is equal to a ring topology of length 4, and is therefore omitted.

Name	Topology	X	Y	Links/router	Tiles	Routers	Links
ring-3	ring	1	3	10	24	3	54
ring-4	ring	1	4	8	24	4	56
ring-6	ring	1	6	6	24	6	60
ring-12	ring	1	12	4	24	12	72
mesh-6	mesh	2	3	6	22	6	58

Table 5.1: The topologies considered.

For each of these topologies, the FFC packing algorithm was chosen to try to map an instance of 100 actors onto an empty system. This was done under degrees of greedy clustering varying from 0% to 60%, in 1% increases. Figure 5.6 shows the percentage of jobs for which a mapping could be found, averaging over 100 instances.

The general trend displayed in this figure is a higher failure rate for topologies which contain a lot of routers. This can be explained through noticing that the network serves as a bottleneck. One of the reasons for this is the choice of topology, combined with the fact that each set of links between two routers has the same capacity as each set of links between a processing tile and a router. The smaller topologies have more processing tiles connected to each router. As with the single-router case, a set of processing tiles connected to the same router have bottlenecks only at the endpoints. This explains the increase of failures once the processing tiles are smeared over more routers.

On the other hand, the individual routers have a cost quadratic in the number of links they connect. For this reason, employing a small number of routers might not be cost-effective. Using Figure 5.6 as a guide, the ring-4

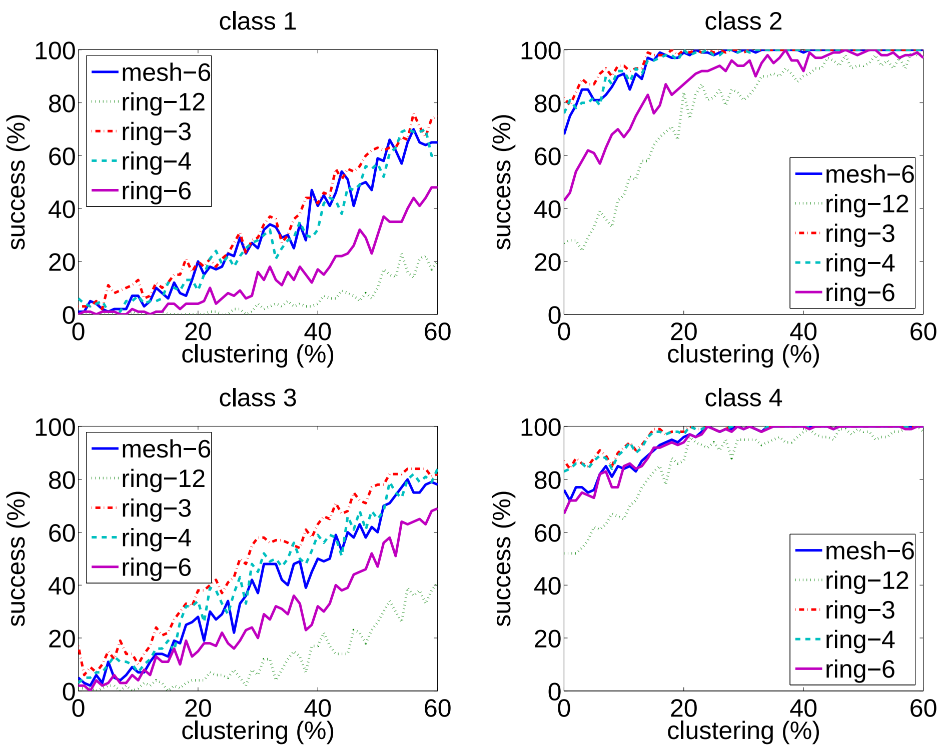

Figure 5.6: The percentage of successful mappings, as a function of the clustering percentage.

topology was chosen as the guideline for further testing. Figure 5.6 pictures this topology as competing with the smallest (ring-3) in terms of failure rate, yet it is probably cheaper to make, due to employing only 8 links per router instead of 10.

5.6.2 Applying FF

First, a simple approach is tested. The packing algorithms FF and FFD are deployed without using any clustering. If these algorithms request more tiles from the system, they will be chosen in a fixed order of increasing distance from the first tile used. The required routes through the network are allocated as soon as both endpoints of a channel are mapped.

This results in a lot of failures, as can be seen in Table 5.2. The single-router case is added as a comparison. This means that it does not suffice to ignore the bandwidth usage of the mapping. In single-router networks, the bandwidth usage only affected the power consumption for the cases tested. Now, it becomes part of the feasibility of the mapping, as the network forms enough of a bottleneck not to allow any bandwidth-unaware mapping to succeed. This means these FF and FFD algorithms are certainly not enough, even though this setup performed well in the single-router case.

This table shows a lot of room for improvement. Subsequent tests will show decent improvement can be made, making this setup a good test case. If there were no failures now, the network is likely to be over-dimensioned, indicating that improved approaches can probably do with less hardware.

Topology	Algorithm	I	II	III	IV
single-router	FF	100	100	100	100
single-router	FFD	100	100	100	100
ring-4	FF	0	0	0	0
ring-4	FFD	0	0	0	0

Table 5.2: The percentage of successful mappings with the FF and FFD algorithms.

5.6.3 Applying Clustering

The FF algorithm can solve the resource allocation problem if the link between a tile and its router is the sole type of bottleneck in the system. This was the case when only a single router exists, but if routers are added, there appear bottlenecks between them of which FF is not aware.

In general, one would expect that if less bandwidth is required in the system in total, these bottlenecks would become less of a problem. The single-router case used clustering to optimise this criterium. By repeating the test in the previous section with the FFC algorithm, the results in Table 5.3 are obtained. This table shows a drop in failures for classes II and IV (which have light edges), but still perform rather badly in classes I and III (which have heavy edges).

Topology	Algorithm	I	II	III	IV
single-router	FFC	100	100	100	100
ring-4	FFC	16	76	3	83

Table 5.3: The percentage of successful mappings with the FFC algorithms.

The indication that clustering helps to protect the packing algorithms from failing, is extended by adding a greedy pre-clustering phase. In Figure 5.7, the result of this is shown for the ring-4 topology[1]. It clearly shows the success rate increasing when more clustering is applied. The FFC algorithm clearly outperforms FF and FFD in terms of this success rate.

[1]Like the tables, the other topologies admit almost identical fail behaviour

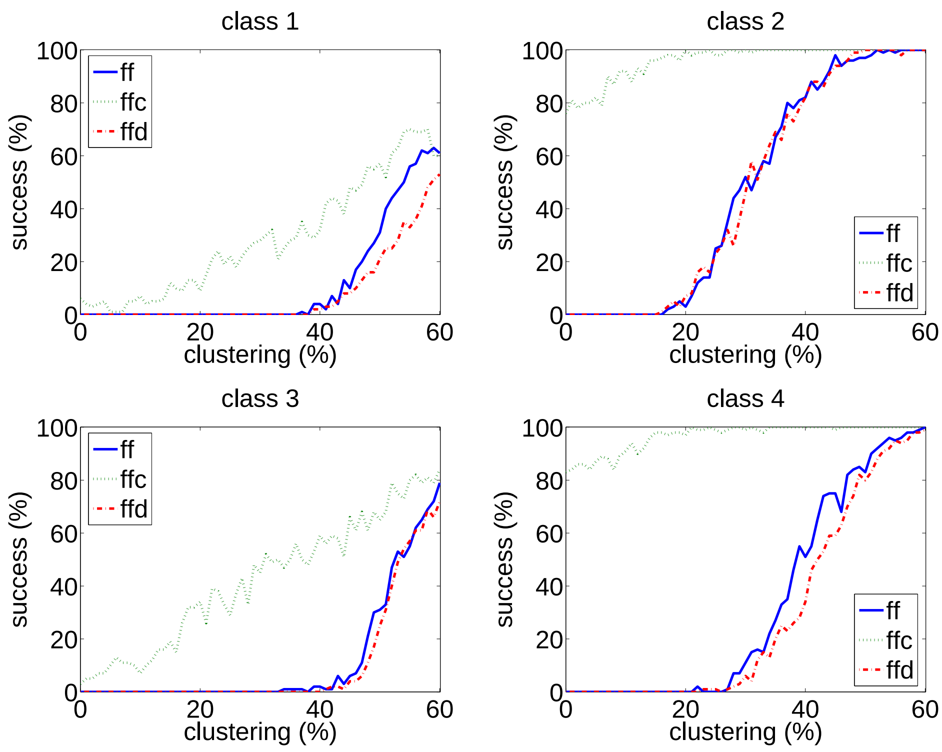

Figure 5.7: The percentage of successful mappings, as a function of the clustering percentage.

This suggests that both forms of clustering can be a key to increase the success rate for packing algorithms, but is not a final solution. For instance, no algorithm could get above a 85% success rate for the instances in classes I and III.

5.6.4 Applying Retries

To further decrease the probability of failure, the RM can try to map the application several times. Since all algorithms are deterministic, different mapping are obtained by shuffling the actors into a random order between mapping attempts. Figure 5.8 shows the gain that can be obtained by doing this. Each line indicates the percentage of successful mapping after a certain number of attempts.

The use of trying to map several permutations of the same job graph is clearly visible, throughout all clustering percentages.

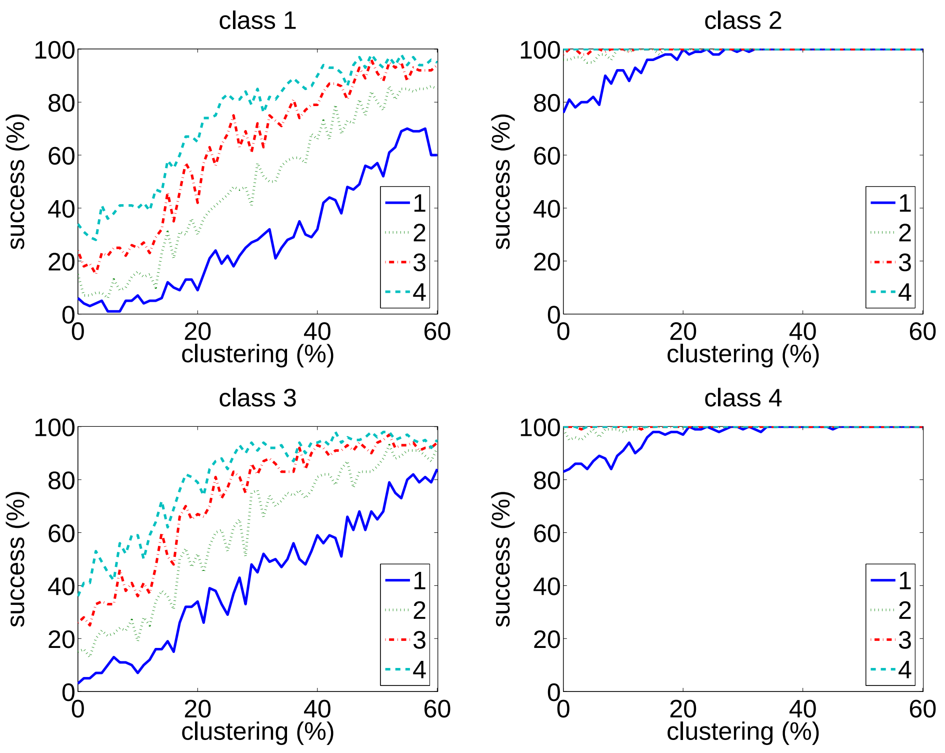

Figure 5.8: The percentage of successful mappings, as a function of the clustering percentage. Each line represents the number of permutations tried.

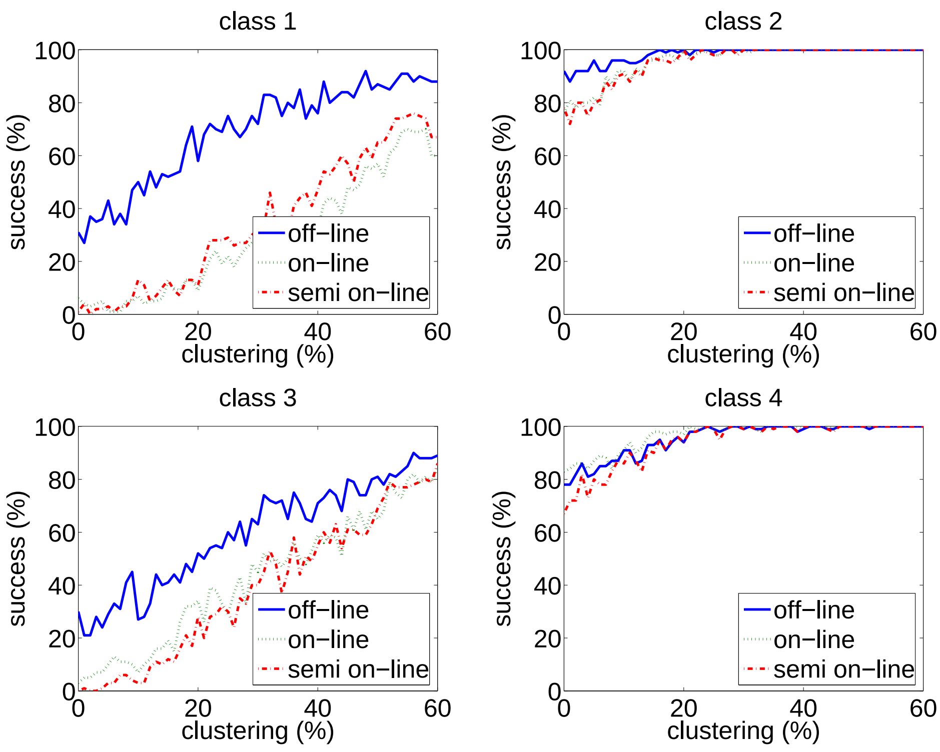

Figure 5.9: The percentage of successful mappings, as a function of the clustering percentage. Each line represents a virtual tile placement algorithm.

5.6.5 Virtual Tile Placement

A weakness of the FFC algorithm, even when combined with clustering, lies in the way it chooses which processing tiles to use. In the previous tests, each time FFC required a new tile, the RM would supply the next one in a predefined order. Section 5.3 called this the *on-line* approach, and acknowledged more sophisticated approaches were possible. The performance, in terms of mapping success rates, of three approaches presented (on-line, semi on-line and off-line), is shown in Figure 5.9. The off-line approach shows a clear improvement over the other two, making it a valid alternative to trying multiple permutations to reduce the failure rate.

The different virtual tile placement algorithms work by trying to reduce the length of the routes required through the network. This effect is measured and displayed in Figure 5.10. It shows the number of links used by each placement algorithm, relative to the original on-line approach. The averages are taken over all *successful* mappings. Since the number of successful mappings differs per approach, these figures should be read with caution. In classes II and IV, almost all mappings were successful for all approaches,

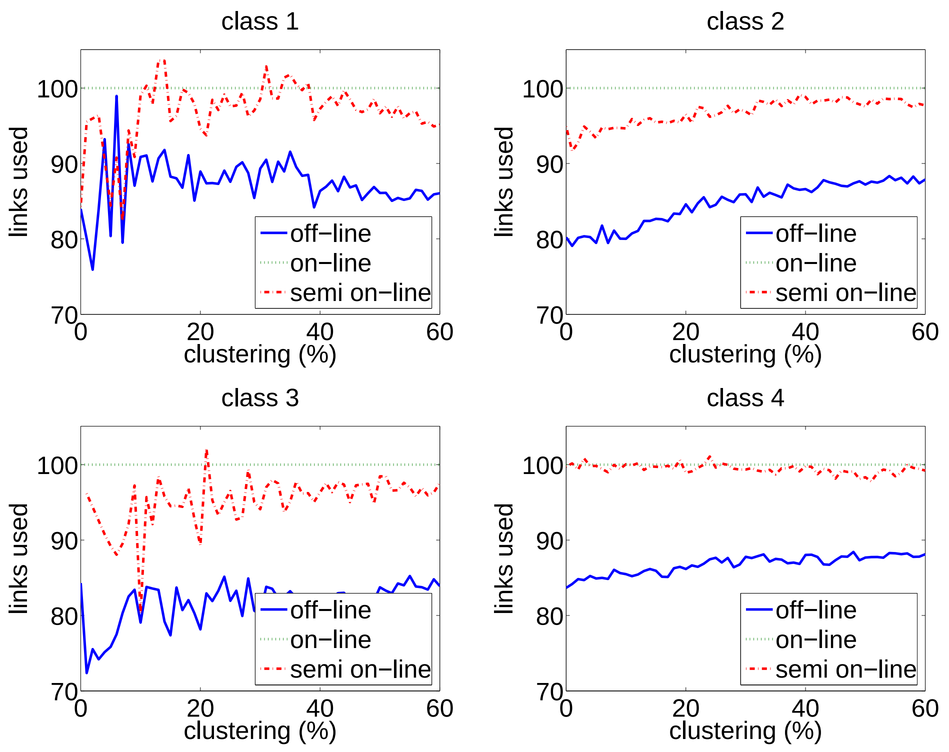

Figure 5.10: The percentage of links used, as a function of the clustering percentage.

making these two the easiest to read.

A decrease of link usage when the off-line approach is chosen, is clearly visible. In classes II and IV, applying the off-line approach yields a 15% profit in terms of link usage. This directly implies profit in terms of power consumption and amount of free resources left for other jobs.

5.6.6 The Game

The stress test game is run using the same parameters as in the tests for single-router networks. It is run for 10000 rounds, using FFC to pack the actors. For each instance, the FFC was allowed to try to map three different permutations. The on-line tile ordering approach was used. The greedy clustering method was employed at 0%, 20%, 40% and 60%, resulting in the success rates shown in Figure 5.11.

The success rate starts out much lower than in the single-router case, as the empty system tests already indicated. The effect of clustering remains the same: it hurts performance. Without clustering, the RM is again capable of delivering a near 100% success rate when a slack of 3–7%, is allowed,

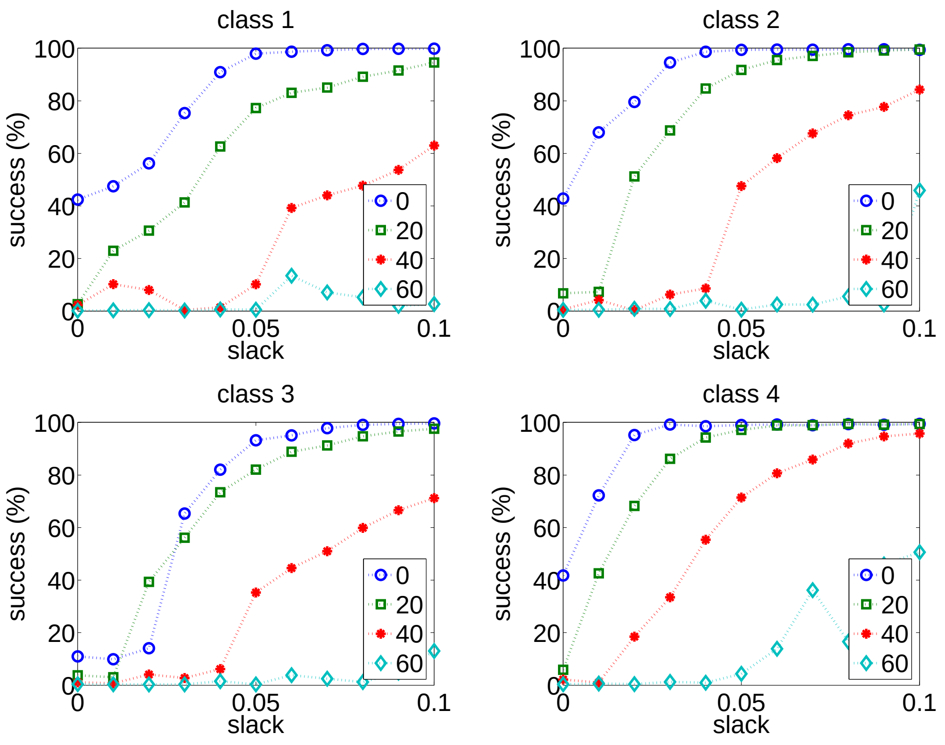

Figure 5.11: The percentage of successful mappings, as a function of the allowed slack. Each line represents a clustering percentage.

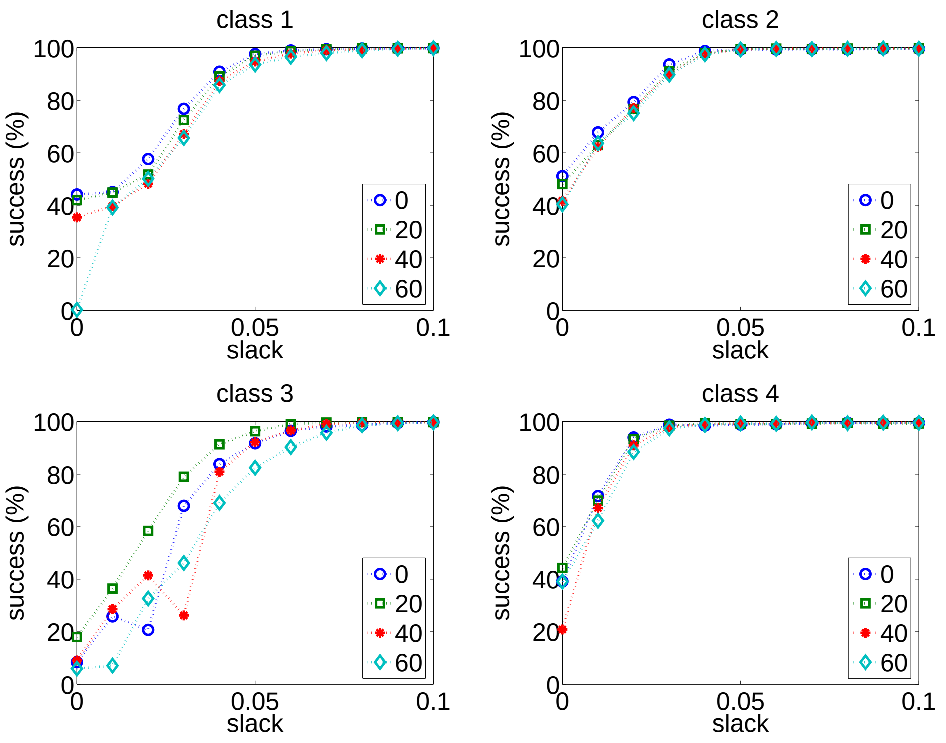

Figure 5.12: The percentage of successful mappings, as a function of the allowed slack. Each line represents a clustering percentage.

depending on the class of jobs.

The tests in the single-router case showed that to increase the performance of the application of the greedy clustering algorithm, a fall back to allow the mapping of an unclustered version of a job works well. For multiple router networks, the result of this is shown in Figure 5.12. Again the performance of 0%-clustering is matched, and even exceeded in some cases.

Finally, the bandwidth usage per channel is scaled up by a factor between 1 and 3. The power of employing CBP is again evident under these circumstances, as can be seen in Figure 5.13. Without CBP, the performance of the RM drops, while the solutions employing CBP manage to keep up the success rate to a far better extend. These results are similar as what was observed in the single-router case, except that the optimal amount of clustering is no longer at or near 60%.

5.7 Conclusions

The mapping of large jobs (of 100 actors) and the addition of a network introduced problems for the RM. The use of the FF and FFD algorithms

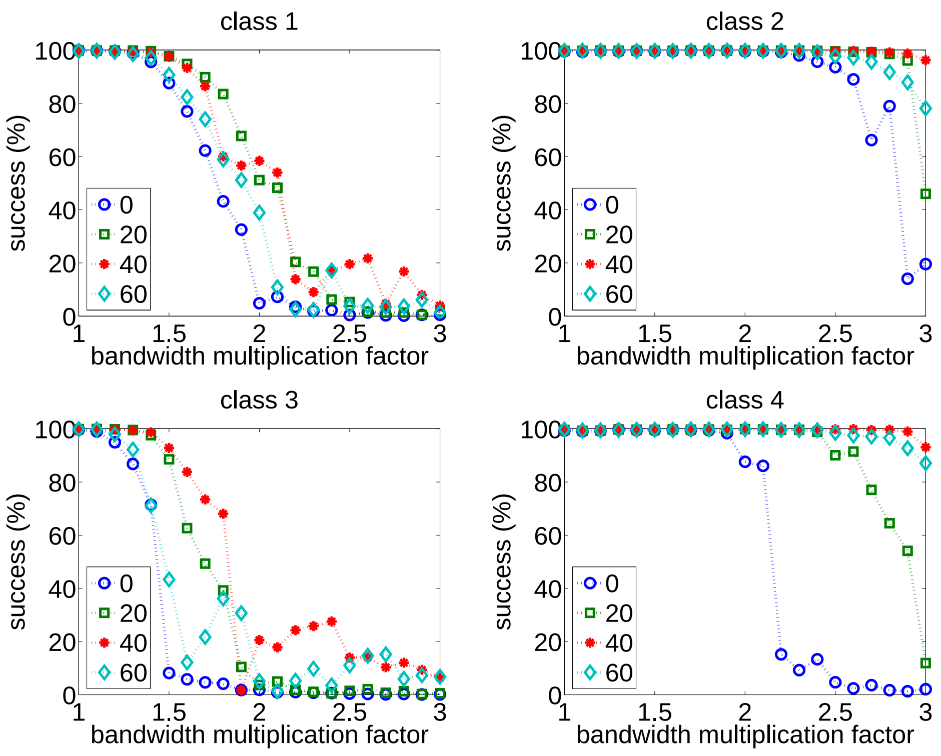

Figure 5.13: The percentage of successful mappings, against a multiplication of the bandwidth requirements. Each line represents a clustering percentage.

failed in all cases, unless a form of clustering was added. Even then, failure rates of 20% were observed. To further increase the chance of success, it was found beneficial to let the RM try to map several permutations of the actor set of a job. Further reductions in resource usage were obtained by employing a smarter distribution of the used tiles. This was accomplished by selecting groups of actors to be mapped on a set of tiles sharing the same router, thus sharing the same bottleneck to the system.

For small jobs, the stress test game returned similar results as for single-router networks. The best approach is again to calculate a clustered solution at design time, and try to map that solution first. When this fails, the RM tries to map the original, unclustered job graph. In many cases, allowing a 10% slack is more than adequate for letting the RM acquire a near 100% success rate in the game. However, when bandwidth becomes a scarce resource, the balance between the aggressiveness of clustering in this pre-calculated solution, and the success rate of the RM is more subtle than in single-router networks.

Chapter 6

Conclusions

This report has analysed a resource allocation problem for homogeneous multiprocessor systems consisting of 10–30 processors. The systems considered are running hard real-time streaming media jobs, which consist of functional components (actors) with data dependencies (channels). The arrivals and departures of these jobs are controlled by an external source. For each arriving job, adequate resources have to be found and allocated at run time.

In this report we consider two system topologies for our performance tests. A small system of up to ten processors can be constructed by connecting them through a single router. For larger systems, a small network with multiple routers is needed. Because problem of resource allocation is NP-complete for both topologies, we devised heuristics to look for a solution. These heuristics were tested by using them to map a single job to an empty system, and we measured the fraction of failed attempts as well as the resource usage. We also stress tested the system by measuring the ability of starting jobs in a system under heavy load. Unfortunately, no realistic job graphs were available for our tests, and so we generated a set of synthetic jobs, with each job having characteristics that were deemed realistic.

In the single-router case, our heuristic is based on trying to map the actors on as few processors as possible. This is augmented by trying to map communicating actors on the same processor, to reduce the bandwidth usage of the packing created. In addition, a packing for an empty system was computed at design time, with good bandwidth and processor usage. Although using this solution uses less resources, it is less flexible to pack at run time because it was created with no other running jobs in mind.

The results in the single-router case show that the use of this precomputed packing result in a 60–80% reduction of bandwidth usage in the single-job tests. In the stress test, using only this pre-computed solution resulted in a high failure rate. This was solved by allowing the algorithm to try to map the original set of components if it failed to map the pre-computed solution. This two-step approach combines the low bandwidth usage of the pre-computed packing with the success rate of mapping the original set. In the cases tested, the heuristic is capable of using 95% of the computational resources available in the system with a near zero chance of failing to map a job.

In the multiple-router case, our heuristic was extended to choose which processors should be occupied by the job, taking into account some basic characteristics of the network. The actors were packed to virtual processors without looking for routes through the network. These virtual processors were then mapped to real ones, placing channel endpoints as close to each other as possible.

The results in the multiple-router case show that for large jobs the use of a pre-computed packing is important to get to a successful mapping. However, for jobs with heavy communications this was not enough to obtain an acceptable success rate. A solution was found in retrying the heuristics on multiple permutations of the input. In addition, the use and placement of virtual processors increased the success rate and showed a reduction of 10–20% of bandwidth usage. In the stress test the heuristics allow a behaviour similar to that of the single router case.

Although the results seem promising, they are obtained using synthetic jobs, and have still to be validated with real jobs. Once this has been done, the model and the performance tests can be extended to heterogeneous systems. Then the problem can be tackled for a wide range of systems covered by the Hijdra architectural template.

Bibliography

[1] Marco Bekooij, Orlando Moreira, Peper Poplavko, Bart Mesman, Jef van Meerbergen, Marc Duranton, and Liesbeth Steffens. "Predictable Embedded Multiprocessor System Design", *Proceedings on CD of Philips Conference on DSP*, 2003.

[2] Thang Bui, Chistopher Heigham, Curt Jones, and Tom Leighton. "Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms", *Proceedings of the 26th ACM/IEEE conference on Design automation*, pp. 775–778, 1989.

[3] Alberto Caprara and Paolo Toth. "Lower Bounds and Algorithms for the 2-Dimensional Vector Packing Problem", *Discrete Applied Mathematics*, vol. 111, pp. 231–262, 2001.

[4] Chandra Chekuri and Sanjeev Khanna. "On Multi-dimensional Packing Problems", *Proceedings of the tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 185–194, 1999.

[5] E.G. Coffman, Jr., C. Courcoubetis, M.R. Garey, D.S. Johnson, P.W. Shor, R.R. Weber, and M. Yannakakis. "Bin Packing with Discrete Item Sizes, Part I: Perfect Packing Theorems and the Average Case Behavior of Optimal Packings", *SIAM Journal on Discrete Mathematics*, vol. 13, pp. 384–402, 2000.

[6] Shantanu Dutt. "New faster Kernighan-Lin-type graph-partitioning algorithms", *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pp. 370–377, 1993.

[7] W. Fernandez de la Vega and G.S. Lueker. "Bin Packing can be Solved Within $1 + \epsilon$ in Linear Time", *Combinatorica 1(4)*, pp. 349–355, 1981.

[8] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. "Networks on Silicon: Combining Best-Effort and Guaranteed Services", *Proceedings of the Design, Automation and Test Conference*, pp. 423–427, 2002. `http://www.dcs.ed.ac.uk/home/kgg/2002-date.ps`

[9] B.W. Kernighan and S. Lin ”An efficient heuristic procedure for partitioning graphs”, *The Bell System Technical Journal*, pp. 291–307, 1970.

[10] Stavros G. Kolliopoulos and Clifford Stein. ”Approximating Disjoint-Path Problems Using Greedy Algorithms and Packing Integer Programs”, *Proceedings of the IPCO Conference on Integer Programming and Combinatorial Optimization*, pp. 153–168, 1998.

[11] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, Springer-Verlag Berlin Heidelberg, 2002.

[12] Edward A. Lee and David G. Messerschmitt. ”Synchronous Data Flow”, *Proceedings of the IEEE*, vol. 75, nr. 9, pp. 1235–1245, 1987.

[13] Silvano Martello and Paolo Toth. ”Lower Bounds and Reduction Procedures for the Bin Packing Problem”, *Discrete Applied Mathematics*, vol. 28, pp. 59–70, 1990.

[14] Jan David Mol. *Resource Allocation for Periodic Hard Real-Time Tasks in Embedded Multiprocessor Systems: A literature study*, Delft University of Technology, 2004.

[15] Gerhard J. Woeginger. ”There is no asymptotic PTAS for two-dimensional vector packing”, *Information Processing Letters*, vol. 64, no. 6, pp. 293–297, 1997.

Appendix A

Notational Index

B	total available bandwidth (in number of links)
$b(e)$	bandwidth requirement of channel e (in number of links)
BB	Branch-and-Bound
BF	Best Fit
BFD	Best Fit Decreasing
CA	communication assist
CAP	Clustering After Packing
CBP	Clustering Before Packing
CDP	Clustering During Packing
$c(e)$	resource requirements of channel e on the tile t hosting the consuming actor
$\delta(e)$	reduction of resource requirements if channel e is internally mapped onto tile t
$d(e)$	number of delays on channel e
E	channels
$F(t)$	vector of available resources on tile t
FF	First Fit
FFD	First Fit Decreasing
$G(V,E)$	an SDF graph G consisting of actors V and channels E
θ	desired sustainable throughput
Θ	maximum sustainable throughput
$i(e)$	resource requirements of channel e on the tile t hosting both endpoints
JNM	Job-to-Network Mapping
KL	Kernighan-Lin bisection algorithm
L	links
MCM	Maximum Cycle Mean
μ_C	the mean of cycle C
μ_G	Maximum Cycle Mean of graph G
μ_θ	a desired upper-bound on the MCM

$\mathbb{N}$	non-negative integers
NI	network interface
$p(e)$	resource requirements of channel e on the tile t hosting the producing actor
R	routers
RD	Route Discovery
$\mathbb{R}$	real numbers
$s(a)$	resource requirements of actor a, excluding its channels, when mapped onto tile t
σ	slack
T	tiles
V	actors
$w(a)$	resource requirements of actor a, including its channels, if it is the only actor on tile t

Appendix B

Proofs

B.1 Clustering Before Packing

The proof of Theorem 1.

Proof. Membership of CBP in NP can be easily shown: given a partitioning, checking if each partition has a total size less than F as well as the bandwidth used by the cut can be calculated in polynomial time.

To prove CBP is also NP-hard, we will reduce the Partition Problem to CBP. It is defined as:

Problem 9 (Partition Problem). *Given a set of items A with each $a \in A$ having a size $s(a) > 0$, is it possible to partition this set into two subsets X and Y such that $\sum_{x \in X} s(x) = \sum_{y \in Y} s(y)$?*

This problem is NP-complete [11]. From a Partition instance (A, s), create the following CBP problem:

$$\begin{aligned} V &= A \\ \forall a \in V : w(v) &= s(v) \\ E &= \{(a,b) | a \in V \wedge b \in V \wedge a \neq b\} \\ \forall e \in E : \delta(e) &= 0 \\ \forall e = (a,b) \in E : b(e) &= s(a) + s(b) \\ F &= \frac{1}{2} \sum_{v \in V} w(v). \end{aligned}$$

In other words, every item of the Partition instance is represented by a vertex in CBP. All vertex-pairs are connected by an edge with a length equal to the sum of the weights of its endpoints.

Now, using the fact that for any set A and function $w : A \rightarrow \mathbb{N}$,

$$\begin{aligned}
\sum_{\substack{x,y\in A\\ x\neq y}} (w(x)+w(y)) &= \sum_{x\in A}\sum_{\substack{y\in A\\ x\neq y}}(w(x)+w(y))\\
&= \sum_{x\in A}\left[\left(\sum_{y\in A}(w(x)+w(y))\right)-2w(x)\right]\\
&= (2|A|-2)\sum_{x\in A} w(x),
\end{aligned}$$

and assuming $\forall a \in A : b((a,a)) = 0$, the total length of the cut will be called L, which can be simplified to:

$$\begin{aligned}
L &= \sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{\substack{x\in P_i\\ y\in P_j}} b((x,y))\\
&= \frac{1}{2}\sum_{i=1}^{n}\left(\sum_{j=1}^{n}\sum_{\substack{x\in P_i\\ y\in P_j}} b((x,y)) - \sum_{x,y\in P_i} b((x,y))\right)\\
&= \frac{1}{2}\sum_{x,y\in V} b((x,y)) - \frac{1}{2}\sum_{i=1}^{n}\sum_{x,y\in P_i} b((x,y))\\
&= (|V|-1)\sum_{x\in V} w(x) - \sum_{i=1}^{n}\left[(|P_i|-1)\sum_{x\in P_i} w(x)\right]\\
&= |V|\sum_{x\in V} w(x) - \sum_{i=1}^{n}|P_i|\sum_{x\in P_i} w(x).
\end{aligned}$$

Since the first term of the last equation is constant for the instance, it suffices to focus on minimising the second term. The sum of weights in each bin is at most F. If all bins used are completely filled, the following holds:

$$\begin{aligned}
\sum_{i=1}^{n}|P_i|\sum_{x\in P_i} w(x) &= \sum_{i=1}^{n}|P_i|\cdot F\\
&= |V|\cdot F.
\end{aligned}$$

If not all bins are completely filled, for at least one bin the sum of weights is less than S. Suppose bin P_h is not completely filled. Then the following holds:

$$\begin{aligned}
\sum_{i=1}^{n} |P_i| \sum_{x \in P_i} w(x) &= |P_h| \sum_{x \in P_h} w(x) + \sum_{i=1, i \neq h}^{n} |P_i| \sum_{x \in P_i} w(x) \\
&\leq |P_h| \sum_{x \in P_h} w(x) + \sum_{i=1, i \neq h}^{n} |P_i| \cdot F \\
&= |P_h| \sum_{x \in P_h} w(x) + (|V| - |P_h|) \cdot F \\
&< |P_h| \cdot S + (|V| - |P_h|) \cdot F \\
&= |V| \cdot F.
\end{aligned}$$

Thus, the total intra-partition edge length is highest if all bins are totally filled. If such a configuration is possible, CBP will choose it. Now, two cases can be extinguished:

- The Partition instance is a ”yes” instance. This means there exists a configuration in which CBP can partition into a set P with containing two fully filled partitions. CBP will choose this configuration and yield a cut equal to:

$$L^* = |V| \sum_{x \in V} w(x) - |V| \cdot F.$$

- The Partition instance is a ”no” instance. This means a partitioning of $|P| = 2$ is not possible, so at least one bin will not be fully filled. CBP will have to settle for a cut with size:

$$L^* > |V| \sum_{x \in V} w(x) - |V| \cdot F.$$

So, if CBP can provide a solution having a $|V| \sum_{x \in V} w(x) - |V| \cdot F$ cut size, a valid solution for the Partition Problem exists. $\square$

The packing problem which remains if CBP is applied (i.e., packing the created clusters onto processors) is still NP-complete. To see this, take a task graph without edges. A CBP algorithm does not have to cluster anything because no bandwidth is used. This implies the input for the packing problem can remain unmodified. This allows the Bin Packing Problem (Problem 2) to be reduced to it.

B.2 Clustering After Packing

The proof of Theorem 2.

Proof. The membership of CAP in NP can be easily seen: given a partition assignment, the number of partitions, the total size of each partition as well as the cut size can be verified in polynomial time.

The fact that CAP is also NP-hard will be shown by reducing the Minimum Bisection Problem to CAP. It is defined as follows:

Problem 10 (Minimum Bisection Problem). *Given a graph $G = (V, E)$ with an even number of vertices, and a natural number k, is it possible to partition the nodes into two subsets of equal cardinality, such that the cut that separates these partitions contains at most k edges?*

This problem is NP-complete. From a Minimum Bisection instance $(G(V, E), k)$, create a CAP instance by retaining $G(V, E)$, and:

$$\begin{aligned} \forall v \in V : w(v) &= 1 \\ \forall e \in E : l(e) &= 1 \\ F &= \frac{1}{2}|V| \\ m &= 2 \\ k &= k. \end{aligned}$$

This reduction can be performed in polynomial time. Since a minimal bin packing of such an instance is trivial (requiring always exactly two bins), no information is gained from the Bin Packing solution that could not be recomputed in polynomial time.

The CAP algorithm will check whether there exists a partitioning of V into two sets, such that the cut size between these partitions is at most k. This corresponds to the same answer as the Minimum Bisection Problem is looking for. □

The packing problem preceding CAP is NP-complete, since applying a graph without edges corresponds to the Bin Packing Problem (Problem 2).

B.3 Job-to-Network Mapping

The proof of theorem 3.

Proof. The NP-complete Partition Problem (Problem 9) can be reduced to JNM as follows. Given a Partition instance (A, s), create the following JNM

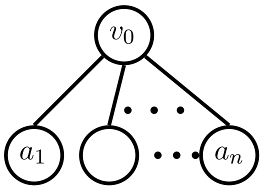

Figure B.1: The Partition instance translated into a JNM graph to be mapped.

instance:

$$\begin{aligned}
V &= A \cup \{v_0\} \\
E &= \{(a,b) | a \in V \wedge b = v_0\} \\
\forall e = (a,b) \in E : b(e) &= s(a) \\
\\
n &= |V| \\
P_1 &= \{p_1, \ldots, p_n\} \\
P_2 &= \{p_{n+1}, \ldots, p_{2n}\} \\
\\
P &= P_1 \cup P_2 \cup \{p_0\} \\
R &= \{q_0, q_1, q_2\} \\
\\
b_0 &= (p_0, q_0) \\
b_1 &= (q_0, q_1) \\
b_2 &= (q_0, q_2) \\
\forall i = 1, \ldots, n : e_i &= (q_1, p_i) \\
\forall i = n+1, \ldots, 2n : e_i &= (q_2, p_i) \\
N &= \{b_0, b_1, b_2\} \cup \{e_1, \ldots, e_{2n}\} \\
c(e) &= \begin{cases} \frac{1}{2}\sum_{a \in A} s(a) & \text{iff } e \in \{b_1, b_2\}. \\ \sum_{a \in A} s(a) & \text{otherwise.} \end{cases}
\end{aligned}$$

This reduction can be performed in polynomial time. The visual representation of this is shown in the Figures B.1 and B.2. For JNM to find a valid mapping, the actor v_0 has to be mapped to p_0, since that tile is the only one capable of outputting $\sum_{a \in A} s(a)$ bandwidth without running into a bottleneck. For the rest of the actors, they have to be split into two sets with an exactly equal bandwidth usage to be able to map all actors onto P/p_0. Whether this is possible is exactly the same question the Partition Problem addresses.

The problem of finding the required routes through the network is in NP. Let a route discovery algorithm be given. A solution candidate for

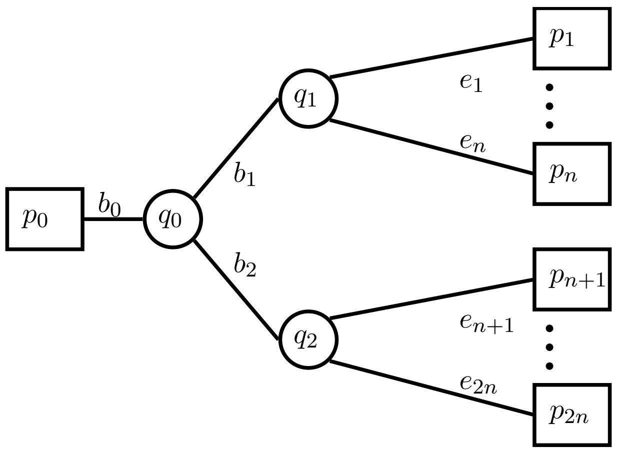

Figure B.2: The JNM network to map the Partition instance on.

the JNM problem can be checked for feasibility in polynomial time once the route discovery algorithm reveals that routing is not a problem. A certificate for the JNM problem can therefore be constructed by encoding a solution candidate, and extending it with the proper certificate for the route discovery problem. This construction yields a polynomial-sized certificate, which can be checked in polynomial time. This implies JNM is in NP, hence it is NP-complete. □