RUN-TIME MAPPING

DYNAMIC RESOURCE ALLOCATION IN EMBEDDED SYSTEMS

TIMON D. TER BRAAK

Members of the dissertation committee:

prof. dr. ir.	G.J.M. Smit	University of Twente (promotor)
dr. ir.	A.B.J. Kokkeler	University of Twente (assistant-promotor)
prof. dr.	J.L. Hurink	University of Twente
prof. dr. ir.	B.R.H.M. Haverkort	University of Twente
prof. dr. ir.	A.A. Basten	Eindhoven University of Technology
prof.	J. Nurmi	Tampere University of Technology
dr.	H. Schurer	Thales Nederland B.V.
prof. dr.	T.T.M. Palstra	University of Twente (chairman and secretary)

UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics and Computer Science, Compute Architecture for Embedded Systems (CAES) group.

This research is conducted within the seventh framework programme (FP7) Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project (IST-215881) supported by the European Commission.

This research is conducted as part of the Sensor Technology Applied in Reconfigurable Systems (STARS) project, funded through FES (Fonds Economische Structuurversterking).

CTIT

CTIT Ph.D. thesis Series No. 16-409
Centre for Telematics and Information Technology
P.O. Box 217
7500 AE Enschede, The Netherlands

Typeset with LaTeX, Ti*k*Z and Vim.
Printed by Gildeprint, The Netherlands.

ISBN 978-90-365-4213-5
ISSN 1381-3617
CTIT Ph.D. thesis Series No. 16-409
DOI 10.3990/1.9789036542135

RUN-TIME MAPPING:

DYNAMIC RESOURCE ALLOCATION IN EMBEDDED SYSTEMS

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. T.T.M. Palstra,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op woensdag 7 december om 11.00 uur

door

Timon David ter Braak

geboren op 2 juli 1984
te Utrecht

Dit proefschrift is goedgekeurd door:

prof. dr. ir.	G.J.M. Smit	(promotor)
dr. ir.	A.B.J. Kokkeler	(assistent promotor)

ISBN 978-90-365-4213-5

Ondanks en dankzij,

Diana, Liz en Sil.

Abstract

Many desired features of computing platforms, such as increased fault tolerance, variable quality of service, and improved energy efficiency, can be achieved by postponing resource management decisions from design-time to run-time.

While multiprocessing has been widespread in embedded systems for quite some time, allocation of (shared) resources is typically done at design-time to meet the constraints of applications. The inherent flexibility of large-scale embedded systems is then reduced to a fixed, static resource allocation derived at design-time. At run-time, unanticipated situations in either the system itself or in its environment may render resources inaccessible that were assumed to be available at design-time. The increased flexibility obtained by run-time resource allocation can be exploited to increase the degree of fault tolerance, quality of service, energy efficiency and to support a higher variability in use-cases. The term *run-time mapping* is used to refer to resource allocation at run-time to meet the dynamic requirements of applications.

A mathematical analysis of the run-time mapping problem shows that each of its subproblems, i.e. task assignment and communication routing, is computationally complex due to the constraints representing the limited resource capacities of the embedded platform. Even if one of these subproblems is solved to optimality, the second optimization problem is still $\mathcal{NP}$-hard. Therefore, two different heuristic techniques are presented to tackle the run-time mapping problem.

The first approach discussed in this thesis is a *deterministic technique*. Both the resources requested by applications, and the resources provided by the platform are modeled as graphs. A divide-and-conquer algorithm exploits the graph structures in order to generate many small resource allocation problems. Each resource allocation problem is a knapsack problem, where the resource requests (the items) are assigned to a subset of the available resources (the bins). In case of an insufficient number of bins, the algorithm increases the set of bins by considering more platform resources, until it runs out of resources. A prototype run-time mapping system which uses this algorithm is evaluated on a many-core processing platform developed in the CRISP project. Multiple real-life applications (various beamforming applications, a GPS receiver, and a dependability monitor) have been tested successfully with the run-time mapper which determines the resource assignment

and configuration. For these applications, the majority of the simulated hardware faults can be circumvented by means of run-time mapping. Empirical evaluation shows, however, that deterministic mapping algorithms do have their weaknesses when it comes to robustness and the ability to provide feedback information. The symmetric structures typically found in both hardware architecture and applications may cause combinatorial searches to spend time evaluating many similar subproblems in a small part of the search space, unable to continue the search in an effective manner. This implies, that the computation time increases vastly without being able to provide a solution or a cause for the failure to provide one.

The second approach discussed in this thesis is a *randomized technique*. Specifically, the meta-heuristic known as *guided local search* is able to improve upon the shortcomings of the first technique. Existing work that applies the guided local search technique to assignment problems only can be used for the task allocation part of our problem. In this thesis, the method is extended to take communication routing into account as well. Guided local search avoids topological orderings of either application or platform graphs. This gives both improvements on robustness in finding solutions and improvements in the quality of feedback information. It is shown that at any time, due to the iterative nature of the method, information can be provided on the relative scarcity of specific resources and the location in the platform that are most critical to the application being mapped. This information may be used for coordination between layers of a hierarchical organized system. Such a system is developed in the context of the STARS project, resulting in a demonstrator consisting of multiple processing boards.

The introduction of full-fledged run-time mapping systems in the domain of embedded systems has long been delayed due to the inherent complexity of the problems to be solved. While similar mapping problems have been solved at design-time for a long time already, different analysis and problem solving techniques are required at run-time. The guided local search technique presented in this thesis provides a balance between robustness and overhead. The results of guided local search and the required computation time on synthetic datasets are competitive with industry standard solvers, while the memory footprint is one or two orders of magnitude lower. Therefore, the algorithm can be implemented on an embedded platform. The computation time required for solving the resource allocation problems at run-time may be further reduced by a *hybrid* form between design-time allocation and run-time adaptation.

Samenvatting

Veel van de gevraagde eigenschappen van computersystemen, zoals een hogere verdraagzaamheid van fouten, het schakelen tussen prestatieniveaus, en een verbeterde efficiëntie wat betreft energieverbruik, kunnen verkregen worden door de keuzes in de toewijzing van resources uit te stellen van ontwerp-tijd tot uitvoeringstijd.

Hoewel het concept waarin meerdere berekeningen (ogenschijnlijk) tegelijkertijd plaatsvinden al een tijd toegepast wordt in geïntegreerde systemen, vindt de toewijzing van (gedeelde) resources nog vaak plaats op ontwerp-tijd om te kunnen voldoen aan alle eisen van applicaties. De inherente flexibiliteit van grootschalige geïntegreerde systemen is dan beperkt door een vastgelegde en statische toewijzing van de resources gemaakt op ontwerp-tijd. Onvoorziene situaties in het systeem zelf of in de omgeving ervan kunnen op uitvoeringstijd er voor zorgen dat bepaalde resources niet meer beschikbaar zijn, terwijl dat op ontwerp-tijd wel zo aangenomen was. De flexibiliteit die verkregen wordt door de toewijzing van resources op uitvoeringstijd te doen kan gebruikt worden om de verdraagzaamheid van fouten te verhogen, de kwaliteit van de service(s) te verbeteren, het energieverbruik te reduceren, en om een hogere variatie in toepassingsmogelijkheden te ondersteunen.

Een wiskundige formulering van het toewijzingsprobleem laat zien dat elk van de deelproblemen, namelijk de toewijzing van taken en de routering van communicatielijnen, reken-technisch gecompliceerd is door de capaciteitsbeperkingen van de resources in het geïntegreerde systeem. Zelfs wanneer een van de deelproblemen optimaal kan worden opgelost, blijft er nog een tweede optimalisatie probleem dat $\mathcal{NP}$-moeilijk is. Dit is de rede dat het probleem wordt aangepakt met twee verschillende heuristische technieken.

De eerste aanpak beschreven in deze dissertatie is een *deterministische techniek*. Zowel de resources gevraagd door applicaties, als wel de resources beschikbaar gemaakt door het systeem worden gemodelleerd in een graaf. Een algoritme met een *verdeel en heers* tactiek benut de structuur van de graaf om het grote probleem op te delen in vele, kleinere toewijzingsproblemen. Elk toewijzingsprobleem is een zogeheten knapzak probleem, waarin de gevraagde resources (de objecten) worden toegewezen aan een deelverzameling van de beschikbare resources (de knapzakken). Wanneer er onvoldoende knapzakken zijn, zal het algoritme, indien mogelijk, het aantal knapzakken verhogen door de verzameling van de beschikbare

resources uit te breiden. Met behulp van een platform met veel rekenkernen, ontworpen en gemaakt in het CRISP project, is een evaluatie gedaan met een systeem dat gebruik maakt van de voorgestelde aanpak. Meerder applicaties zijn succesvol getest met het systeem dat de toewijzing van resources en de bijbehorende configuratie voor zijn rekening neemt; digitale bundelvormers, een GPS ontvanger, en een betrouwbaarheidsmonitor. Met het gebruik van deze applicaties is gebleken dat het overgrote deel van de fouten in de hardware omzeild kan. Echter laat empirische evaluatie zien dat er zwakheden in de aanpak zitten op het gebied van robuustheid en in de mogelijkheden tot terugkoppeling van informatie. De symmetrische structuren kenmerkend in de architectuur van de hardware en in applicaties veroorzaken problemen in de zoekmethodes, waardoor er veel tijd besteed wordt aan de evaluatie van soortgelijke deelproblemen in slechts een klein gedeelte van de zoekruimte, waardoor het onmogelijk wordt om de gehele zoekruimte te bekijken op een effectieve manier. Wanneer dit het geval is, dan neemt de benodigde rekenkracht sterk toe zonder dat er ook maar een oplossing gevonden wordt, of zonder dat er een reden gegeven wordt voor het niet vinden van een oplossing.

De tweede aanpak beschreven in deze dissertatie is een *gerandomiseerde techniek*. Een meta-heuristische methode bekend als *guided local search* biedt mogelijkheden om de tekortkomingen van de eerste aanpak te verhelpen. Bestaand werk waarin deze methode gebruikt wordt kijkt alleen naar het deelprobleem waarin toewijzing van taken opgelost wordt. In dit werk is de methode uitgebreid met de mogelijkheid om de routering van communicatielijnen ook mee te nemen. 'Guided local search' maakt geen gebruik van topologische sorteringen van de applicatie graaf of platform graaf. Dit geeft een verbetering in de robuustheid van het vinden van oplossingen en in de mogelijkheid tot het terugkoppelen van informatie. De iteratieve manier van werken maakt het mogelijk om op elk moment informatie te produceren over de relatieve schaarste van resources en over de locaties in het systeem die het meest kritiek zijn in de toewijzingsprocedure. Deze informatie kan gebruikt worden in de coördinatie tussen verschillende niveaus in een hiërarchisch georganiseerd systeem. Een voorbeeld van zulke systemen is ontworpen in het kader van het STARS project, wat resulteerde in een demonstratie bestaand uit meerdere rekenborden.

De introductie van een volwaardig geïntegreerd systeem dat de toewijzing van resources op uitvoeringstijd doet is lange tijd vooruit geschoven vanwege de complexiteit van de onderliggende problemen. Hoewel soortgelijke problemen al wel opgelost zijn op ontwerp-tijd, zijn er andere analyse en oplossingstechnieken nodig voor de toepassing op uitvoeringstijd. De 'guided local search' methodiek voorgesteld in deze dissertatie biedt een balans tussen robuustheid en overhead. Gebruikmakend van een synthetische dataset, blijken de resultaten van deze methode en de daarbij behorende rekentijd competitief met gevestigde oplossingen; dit, terwijl de vereiste hoeveelheid geheugen een of twee ordergroottes lager is. Hierdoor kan het algoritme toegepast worden in een geïntegreerd systeem. De benodigde rekentijd om het toewijzingsprobleem op uitvoeringstijd op te lossen kan eventueel nog verder gereduceerd worden door gebruik te maken van een *hybride* vorm van resource toewijzing op ontwerp-tijd en aanpassing daarvan op uitvoeringstijd.

Dankwoord

De eerste stappen richting de voltooing van dit proefschrift waren gezet toen ik aanklopte bij de Computer Architecture for Embedded Systems (CAES) vakgroep van de Universiteit Twente. Gerard Smit, de hoogleraar van de vakgroep legde mij een aantal mogelijke afstudeeronderwerpen voor. Nadat die ter plekke op een kladje waren gekrabbeld tijdens een mondelinge toelichting, was er van mijn kant wel wat interesse voor iets dat *run-time mapping* werd genoemd. Omdat dit concept mij nog niet direct duidelijk was, werd mijn afstudeerbegeleider Philip Hölzenspies erbij gehaald. Door de keuze voor dit onderwerp heb ik mijzelf in zeer uiteenlopende disciplines moeten verdiepen. Allereerst wil ik Philip bedanken voor de begeleiding tijdens mijn afstuderen. Het begon stevig met een duidelijk uitleg dat ik er niet met een 'zesje' vanaf zou komen, en dus wel serieus aan de slag zou moeten. Dit was wellicht een onbewust opgelegde druk om zelf ook tot concrete resultaten te komen. In een goede samenwerking is de basis gelegd voor een tweetal proefschriften, elk met een eigen focus en contributie. Van Philip heb ik ook een goede introductie gehad tot het promotietraject, met de bijbehorende uitstapjes naar conferenties en de procedures met betrekking to het publiceren van resultaten. Philip, bedankt voor dit alles.

Tijdens mijn afstuderen en in het begin van mijn promotietraject draaide het CRISP project op volle toeren. In de context van dit project kwam ik in aanraking met het bedrijf Recore Systems, dat samen met Thales en Atmel een many-core DSP platform heeft ontwikkeld. De verantwoordlijkheid van deze bedrijven was vooral hardware georienteerd en processmatig. De CAES groep had de taak om een prototype te maken van het run-time mapping concept. Mede hierdoor kwam er veel verantwoordelijkheid wat betreft het opbouwen van de software bij de groep terecht, veel meer dan slechts het demonstreren van de concepten. Samen met Hermen Toersche heb ik veel praktisch werk verzet en hierbij de benodigde kennis opgedaan. De mooiste momenten waren tijdens onze werkzaamheden in het ESD lab van Thales, waar we zonder al te veel ervaring de eerste LEDjes en de boot procedure van de GSP werkend moesten zien te krijgen, terwijl er regelmatig medewerkers van Thales over onze schouders meekeken van nieuwsgierigheid. Hermen, het werk wat je gedaan hebt in je afstudeerproject was cruciaal voor zowel het CRISP project alswel de demonstratie van de run-time mapping concepten. Ik denk dat zeer weinig mensen je dit na zouden doen; dit zit in zowel de complexiteit, een on-

duidelijke planning door externe afhankelijkheden en de benodigde commitment. Bedankt Hermen voor je inzet en de leuke samenwerking. Niets was teveel en geen vraag te gek. In het STARS project werden de resultaten met CRISP gebruikt om een demonstrator te maken op grotere schaal. Hoewel we niet de resources hadden om het oorspronkelijke plan te realiseren, heb ik toch kunnen bijdragen aan de demonstrator die door Jonathan Melissant en Ruben Marsman gemaakt is. Bedankt voor de extra uitleg over beamforming, de moeite die jullie in de demonstrator hebben gestoken en het begrip voor de koerswijzigingen die we moesten maken.

Op de universiteit waren Anja Niedermeier en Robert de Groote mijn kamergenoten. Onze promotieonderwerpen waren te verschillend om veel inhoudelijke discussies te hebben, maar wat mij betreft hebben we een leuke tijd gehad samen. Voor Anja heb ik bewondering dat ondanks alles haar promotie in het daarvoor gestelde tijdsbestek is afgerond; iets wat vaak een lastige opgave blijkt te zijn. Het sterke punt van Robert is zijn theoretische insteek die hem in staat heeft gesteld, tegen de stroming in, goede resultaten te boeken op een onderwerp die door anderen al als uitgekauwd werd gezien. Mede daardoor kon ik hem meer dan eens plagen met een praktische insteek in het onderwerp. Anja en Robert, bedankt voor onze tijd samen.

Tijdens het gehele traject zijn mijn promotor Gerard Smit en mijn co-promotor André Kokkeler zeer belangrijk geweest. Al in mijn afstudeertraject nam Gerard de moeite mij te begeleiden in het *academieassistenten*-programma, om meer bekend te raken met het wetenschappelijk onderzoek. Naast de kansen die mij werden geboden, wil ik Gerard bedanken voor het vele review werk; met name wanneer de tijdsdruk hoger werd wist hij vaak nog wel een gaatje te vinden om het werk te bekijken. Ook ben ik dankbaar voor het vertrouwen en de steun in de laaste fase van het afronden van dit proefschrift, dat wat langer op zich heeft laten wachten. De grote bijdrage van André is om op een correcte manier toch kritisch commentaar te leveren op de details, zelfs wanneer de inhoud buiten zijn directe expertise ligt. Vele kernwoorden uit dit proefschrift zijn op André van toepassing; abstract denken, het managen van deadlines, een hoge doorvoersnelheid en het voorkomen van chaos. Gerard en André, bedankt voor dit alles. Zijdelings is Johann Hurink vanuit de wiskundige optimalisatie hoek ook betrokken geweest bij het run-time mapping onderwerp; zowel bij het proefschrift van mijn voorganger Philip alswel bij dit proefschrift. De uitgebreide en gedetailleerde feedback alswel het behoud van het overzicht heeft mij erg geholpen. Naast de academische aspecten kan iedereen van de CAES groep altijd bouwen op het uitstekende ondersteunende werk van het secretariaat; Marlous, Thelma en Nicole, bedankt voor het regelen van reispapieren, formulieren en uitjes, en het beantwoorden van alle vragen waarop we zelf het antwoord hadden moeten weten.

Tevens wil ik ook Recore Systems bedanken voor nogal uiteenlopende zaken. Ten eerste voor de openheid en bereidheid om studenten te begeleiden en mee te laten werken aan relevante problemen. Zoals eerder genoemd, ben ik zo tijdens mijn Master's project in aanraking gekomen met het CRISP project en de GSP. Direct na

het aflopen van mijn contract bij de Universiteit Twente ben ik bij Recore Systems aan de slag gegaan. Mijn PhD thesis was nog niet afgerond, maar Recore Systems heeft mij altijd ondersteund en aangemoedigd om dit alsnog af te ronden. Specifiek wil ik Gerard Rauwerda en Kim Sunesen bedanken voor het vertrouwen in mij en voor de welgemeende interesse in de voortgang.

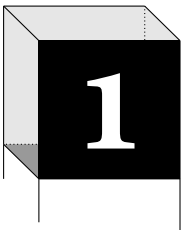
De afronding van dit proefschrift was nooit gelukt zonder de steun van mijn vrouw Diana. Hiervoor eerst een anekdonte. Een opmerking van Gerard over de mogelijke nadelen van de aanpak in Hoofdstuk 5 resulteerde in de vraag hoe ver we met die aanpak van het optimum af zitten. Diana suggereerde om te vermelden dat we "heel dicht bij" zitten. Ik denk dat een beter antwoord een tweede promotietraject vereist. Los van het feit dat dit waarschijnlijk buiten mijn intellectuele vermogens ligt, zal ik dit Diana, Liz en Sil niet aandoen. Diana, dank je wel dat je me gesteund hebt waar nodig, geholpen hebt met elke keer weer een nieuwe planning te maken, en me onder druk gezet hebt om hoofdstukken af te ronden. Liz, ook al ben je momenteel te jong om goed te begrijpen wat ik aan het doen ben geweest, is het altijd leuk om weer thuis te komen; vooral je enthousiast uitroep "papa!". Sil, je bent (momenteel) onze levende knuffelbeer met een grote eigen wil. Ik denk dat jullie mede oorzaak zijn geweest voor enige vertraging van dit proefschrift, maar ik had zeker niet minder tijd met jullie willen doorbrengen.

Timon ter Braak
Enschede, november 2016

Contents

List of Figures

List of Tables



1 Introduction

Abstract – *Computer systems are subject to continuously increasing performance demands. However, energy consumption has become a critical issue, both for high-end large-scale parallel systems, as well as for portable devices. In other words, more work needs to be done in less time, preferably with the same or with a smaller energy budget. Future performance and efficiency goals of computer systems can only be reached with large-scale, heterogeneous architectures. Due to their distributed nature, control software is required to coordinate the parallel execution of applications on such platforms. Abstraction, arbitration and multi-objective optimization are only a subset of the tasks this software has to fulfill. An essential problem in all this is the allocation of platform resources to satisfy the needs of an application.*

General purpose (micro)processors have an instruction set that is tailored towards control-oriented applications, making them very flexible. Techniques like pipelining, out-of-order processing and branch prediction try to minimize the latency per computation. This leads to designs optimized for best-effort processing. A dominant factor regarding performance is the latency of memory accesses. With each generation of processor architectures, attempts are made to increase the memory bandwidth and to decrease the average memory latency. Latency may be mitigated by using a memory hierarchy with caches, or by switching to another thread while waiting for the memory access to complete.

Hide memory latency; wait in parallel:
The Gatling gun was one of the first well-known rapid-fire guns. Other guns simply increased their rate of fire, but quickly found that their gun barrels overheated if they attempted to fire too quickly. The Gatling gun used multiple barrels, each of which individually fired at a slower rate, but when rotated in succession allowed a continuous stream of bullets to be fired while allowing the barrels not in use to cool down. The time it takes for a discharged barrel to cool down is similar to the latency of a memory access [126].

For data processing on a general purpose processor, many control instructions have to be repeated for each data item, resulting in a large overhead and superfluous energy expenditure. However, instead of using hardware for speculative processing, additional functional units could be integrated to increase instruction-level parallelism. The programmer or compiler then provides information about the grouping and mapping of instructions to the available functional units, for example, using very large instruction words (VLIWs). This more fine-grained control over the hardware units inside a processor potentially leads to increased throughput and more (energy) efficient processing. Having a single, but larger, instruction operating on bigger data items makes VLIW cores more suitable for digital signal processing. While modern digital signal processors (DSPs) support most operations found in general purpose processors (GPPs), the mechanisms that implement the instruction set are throughput-oriented. Control instructions may therefore vastly decrease the performance of a DSP, as it lacks the hardware to hide the induced latency.

For some functionality, it is beneficial in terms of computational performance and energy efficiency to implement a function in dedicated hardware; a so-called hardware accelerator (HWA) or application specific integrated circuit (ASIC). Creating such a dedicated piece of hardware allows a designer to make use of designs optimized for a specific function, rather than making flexibility trade-offs and/or adding various general purpose support structures. Table 1.1 illustrates the gain in energy efficiency by using specialized hardware, using a 1024-point (radix-4) fast Fourier transform (FFT) as benchmark. Note that the data of Table 1.1 may contain inaccuracies or unfair comparisons, due to variations in process technology, in clock speed, in applied voltage, in number of cores and in measurement conditions. Still, the numbers presented in Table 1.1 contribute to the case of heterogeneous computing.

10x10 optimization:
Through two decades of rapid performance improvement, the dominant optimization paradigm has been 90/10, which focuses on 90% of the workload and on optimizing the activities of that dominant portion to optimize a general-purpose architecture. However, technology scaling and architecture trends do not favor investment of hardware resources in a general-purpose, high performance core [18].
...
In this world, 90/10 optimization no longer applies. Instead, optimizing with an accelerator for a 10% case, then another for a different 10% case, then another 10% can often produce a system with better overall energy efficiency and performance. We call this '10x10 optimization', as the goal is to attack performance as a set of 10% optimization opportunities – a different way of thinking about transistor cost, operating the chip with 10% of the transistors active – 90% inactive, but a different 10% at each point in time [13].

Table 1.1: Energy efficiency of various architectures for a 1024-point (radix-4) FFT [53, 80, 87, 90, 97, 100, 131].

Platform	Time (μs)	Power (Watt)	Energy / FFT (μJ)
GPP:			
Intel Pentium 4 @ 3 GHz	23.9	52	1250.0
Intel Xeon (2 cores) @ 3 GHz	1.8	95	171.0
Intel Nehalem (4 cores) @ 3.2 GHz	1.2	130	156.0
ARM 920T @ 250 MHz	106.67	0.06	6.67
GPU:			
nVidia Tesla C1060	0.3	188	56.4
nVidia Tesla C2070	0.2	225	36.0
FPGA:			
XilinX XC2VP2 @ 63 MHz	20.32	0.83	16.84
Altera Stratix @ 275 MHz	4.7	0.88	4.1
DSP:			
TI C55x @ 100 MHz	277.2	0.06	17.0
TI C55x @ 60 MHz	462.0	0.02	11.1
TI C6416 @ 720 MHz	8.34	1.19	10.0
TI C6678 (8 cores) @ 1.2 GHz	0.9	10	8.6
Xentium @ 200 MHz	23.4	0.03	0.7
ADSP-21262 @ 200 MHz	46.0	0.63	29.0
ASIC:			
TI C55x HWA FFT @ 100 MHz	73.2	0.04	2.8
FASRA @ 120 MHz	42.8	0.05	1.94
TI C55x HWA FFT @ 60 MHz	121.9	0.01	1.8
FFTTA @ 250 MHz	20.9	0.08	1.6
ISSCC 2011 0.27V FFT @ 30 MHz	4.3	0.004	0.02

1.1 Enlarging the toolbox: heterogeneous computing attempts to regain flexibility

The need for energy-efficient architectures is not only driven by cost. Another aspect is, that Dennard scaling[1] [27] has come to an end; improvements in process technology still allows us to put an increasing amount of transistors on the same chip area, but we can no longer reduce the voltage to operate them reliably. The power consumption of potential designs that could fit onto a chip exceeds common power envelopes by one or two orders of magnitude [13]. Instead of a few high-performance general purpose cores, we need to put transistors into heterogeneous processing units, each specialized for a different set of functions. This paves the way for the '10x10 optimization' design approach, which advocates a set of small (10%) optimizations for various cases over one big optimization for the average case.

[1] The power density of transistors stays constant; a reduction in transistor size is a reduction in power consumption.

Heterogeneous architectures are then defined by a parallel composition of domain specific processing elements and general purpose processors. By using only a subset of the components at a time, the available energy is distributed over space and time. The phenomenon that parts of a chip are disabled due to a limited energy budget is known as *dark silicon*. By delivering more performance per Watt [13, 75], heterogeneous computing will become the industry standard for developing new architectures for a variety of energy-aware application domains.

The composition of many specialized blocks yields bigger architectures (using the increasing transistor budget of technology scaling), and brings more flexibility, at the cost of increasing the complexity of these systems. Unfortunately, few applications are suitable for *strong scaling*, which is the concept of applying more (specialized hardware) resources to the same problem size to get results faster [98]. Amdahl [4] identified this problem by observing that the sequential part of a problem is the limiting factor for the potential speedup of an application. Gustafson [45] has shed a different light on the matter by stating that applications will increase the total amount of computation to benefit from the facilities provided by computer architectures. This is known as *weak scaling*.

Amdahl's law:
For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit co-operative solution…
The nature of this overhead (in parallelism) appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate, even if the housekeeping were done in a separate processor…
At any point in time it is difficult to foresee how the previous bottlenecks in a sequential computer will be effectively overcome [4].

Gustafson's law:
It may be most realistic to assume run time*, not* problem size*, is constant. Hence, the amount of work that can be done in parallel varies linearly with the number of processors [45].*

Large-scale heterogeneous architectures enable both strong and weak scaling of applications that solve computational problems of sufficient scale. The key challenge is to find a high-quality projection (mapping) of the application onto the various hardware components. Performance and energy efficiency is then gained by matching the right functionality to suitable specialized processing units. Inevitably, arbitration and control of the hardware is required to enable resource sharing in order

to fully exploit the efficiency of the platform. Some hardware components need to be configured for multiple, different contexts. Some processing elements are capable of managing the context themselves, while others need a host processor to configure and switch contexts. The degree of autonomy of a reconfigurable block is best described in terms of its coupling to another processor. We identify functional units, co-processors, attached processing units and standalone processing units. Figure 1.1 shows various degrees of coupling to these units from the perspective of a host processor. Interaction between these units takes place by reading and writing the contents of various memories and registers, and transferring that data through the interconnect. The distributed nature of such a system requires us to think not only of the capabilities of processing units, but their location as well (considering communication overhead). This is further elaborated in the next section.

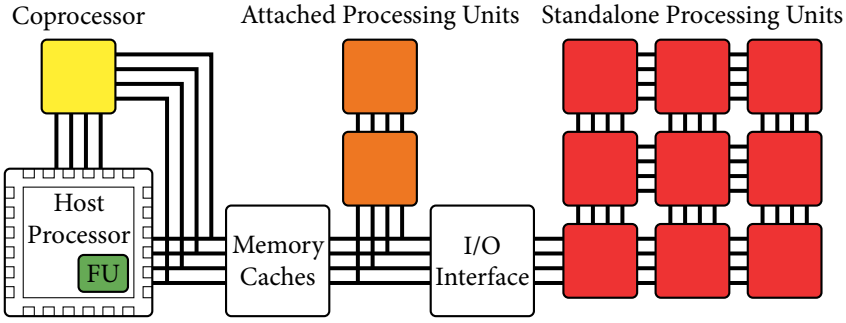


Figure 1.1: Degree of coupling between reconfigurable logic and the host processor [23].

1.1.1 Distributed Memory Systems

Its clear from Figure 1.1 that the impact of data exchanged between two processing units relates to the organization of the (hierarchical) memory structure within an architecture, to the interconnect between processing elements and to the operational use of the system. Moreover, heterogeneous systems often have a complicated, distributed memory structure to sustain a certain performance level with a low energy budget. Amdahl's laws [3] state that architectures should support sufficient memory and input / output (I/O) capabilities to keep up with the computations.

Amdahl's balanced system law and memory law:
1 byte of memory and 1 byte per second of I/O are required for each instruction per second supported by a computer [3].

Although being rules-of-thumb, and the real numbers are very application dependent, the laws comply with observations that today's processors are spending a

significant part of their time waiting for I/O and memory. The cost reduction of transistors allows for larger local memories and register files, reducing the latency and associated energy consumption of data transfers. While improving the *locality of references*, the complexity of programming such systems is increased.

Next to heterogeneous processing, the energy consumption of a computation may be further reduced by voltage (V) and frequency (f) scaling techniques [13], and by lowering the (average) activity (α). In general, the energy required for a bit flip can be characterized by the dynamic power dissipation P of complementary metal-oxide-semiconductor (CMOS) technology with $P = \alpha CV^2 f$. Unfortunately, the energy required for data transfers does not scale accordingly. Larger chip designs incorporate more chip wire, adding up the capacitance C required to charge the circuits. For example, the Intel Broadwell 14nm chip has 13 metal layers to connect all its features and areas [67]. Optimizing data locality will, therefore, have a major impact on energy reduction in future systems [13, 98]. Figure 1.2 shows the significance of *data locality* with respect to energy consumption, measured and/or estimated by [26]. Relative to a double-precision floating point operation (on-chip), reading from an external dynamic random-access memory (DRAM) chip is about 800 times more expensive. In this thesis we consider energy reduction through heterogeneous processing as well as data locality.

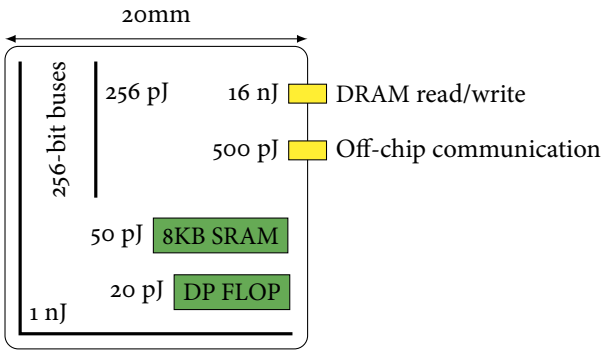


Figure 1.2: Estimated energy usage of a chip in 40nm CMOS technology.

1.2 Programmability of heterogeneous distributed systems

Whereas heterogeneous processing and distributed memory address the energy efficiency problem, the programmability of these platforms is far from trivial. A long-standing challenge is not to design individual efficient hardware components, but to design an aggregation of components (including processing, memory and interconnect), and especially to define a programming model for those architectures [98]. It often requires extensive knowledge and effort to program heterogeneous systems effectively due to the multiple levels of potential parallelism. A software engineer may have to handle concurrent processing and inter-processor

communication explicitly, and has to master hardware specific programming interfaces and tool-kits. One of the most complex platforms ever made are multi-user, heterogeneous clusters made up of processors of different architectures, interconnected via a heterogeneous communication network [29, 98]. Programming these platforms is a big challenge. Therefore, a programming model needs to provide the programmer with the desired abstraction, while at the same time, sufficient information about the programmers intention and associated requirements has to be provided to the underlying system [96]. The goal of numerous research projects is to improve the state-of-the-art on such programming models. The projects funded through the European FP7-ICT Programme alone are already too many to list here, but the author of this thesis was involved in a number of them [61, 62, 63, 64, 65].

One approach to tackle the programming problem is to reduce the design complexity by breaking the software into well-defined pieces. In this approach, multithreading is the dominant paradigm of concurrency in which the 'pieces' are known as *threads* that can execute concurrently. Communication between threads commonly occurs through writes in shared memory regions. The threading model is often employed in complex computing systems with non-deterministic concurrency [68]. The executing thread may be arbitrarily interrupted, after which another thread begins or resumes execution. *Critical sections* must be guarded by explicit concurrency control, commonly implemented with locks. However, static analysis of such shared memory concurrent programs is undecidable in general [16], and as a consequence of these properties, composability of threads is limited [48].

> *Things that are composable are good because they enable abstractions, meaning that they enable us to reason about code without having to care about all the details, and that reduces the cognitive burden on the programmer.*
>
> *– Anonymous user, 2010*

Implicit communication and data sharing between threads may be made explicit by adoption of the paradigm of *message passing*. Separating communication from computation allows engineers to develop *computational kernels* in a conventional and sequential manner. Two kernels may then be assembled together if the message format used on their communication channels is compatible. The internal working of these kernels is not relevant to other kernels, as opposed to lock-based multithreading. These individual kernels may then be used to compose larger systems.

> *Concurrency is a way to structure a program by breaking it into pieces that can be executed independently. Communication is the means to coordinate the independent executions [54].*
>
> *– Rob Pike, 2012*

The message passing paradigm matches the streaming applications domain and the clear separation between computation and communication suits explicit resource management. Therefore, in this work, the message passing paradigm is adopted. Applications are modeled as a composition of (processing) *tasks* that exchange data over *communication channels*. The *explicit communication* between tasks and between their associated resources may be exploited by a coordination layer that abstracts from specific configurations of a system. The coordination layer then provides guarantees to an application about the resource availability and substantiates resource claims on the hardware level. The message passing paradigm provides flexibility with respect to the location of those resources without explicit knowledge on those locations at the application-level.

> *Any general purpose, configurable system requires queues between functions to adjust for different processing rates of the various units. The control of queues is a function of the system resource manager, as data is one of the system resources. A means to connect modules to form programs and to assign data is needed [30].*
>
> *– N.P. Edwards, 1977*

1.2.1 Reservation-based resource partitioning

The task of deciding how to allocate resources to competing users is known as *scheduling*. To arbitrate over a resource, we need to know which users are waiting for the resource. In the theory of communicating sequential processes [54], each resource is modeled as a process. Whenever access to a resource is requested, a new *virtual* resource is acquired. The virtual resource interacts with the real resource whenever required, but, the actual communication pattern is *concealed* from the user. The paradigm of actual and virtual resources is very important in the design of resource-sharing systems [54]. It provides a clean interface to the user, and it guarantees a disciplined access to the actual resource. Acquisition of a resource is, therefore, not an atomic action; it must be split into two events; a *please* and a *thankyou*. Arbitration is then performed over the aggregation of all users that compete for the actual resource (said a *please*). The system should then guarantee resource access to all the users that received a *thankyou*. This concept is known as *reservation-based resource partitioning*, which is acknowledged as a paradigm to design real-time systems [1, 107]. In this paradigm, a resource management policy [74, 81] should ensure that the system remains in a correct state, where:

- an application is only allowed to start if the system can allocate, upon request, the resource budget it requires to meet its performance constraints [*admission control*],
- the access of an admitted task to its allocated resources cannot be denied by any other task [*guaranteed resource provision*].

Applications reserve not just a single virtual resource, but rather request a subset of the systems resources, which may be considered to be a complete virtual system [107]. The ability to define these virtual systems, thus, relies on combined mechanisms for admission control (at design-time or at run-time), allocation or scheduling (at design-time or at run-time), accounting (at run-time), and enforcement (at run-time) [107]. This requires the software to be *location transparent*; that is, to be free of hidden assumptions on where and how it will be executed.

1.2.2 Run-time mapping

The number and 'shape' of virtual platforms depends on the applications that need to be executed. The resource allocation has to be performed at run-time, in case the available hardware is not statically defined at design-time and/or subject to change, or when the set of applications, the possible combinations, or their specific I/O ports are not predefined. A run-time resource manager then has to match the *resource demand* of applications with the *resource provision* of a platform. What resources are required during execution should be specified per application. This resource demand specification can be used to allocate sufficient platform resources to the application, and to reconfigure the system accordingly. Within an application, tasks exchange data with each other through communication channels, which have to provide enough bandwidth with a bounded latency to sustain the required performance. So, not only the amount and type of resources required matters, but the location of those resources as well. This spatial factor increases the complexity of the resource allocation problem, making established scheduling algorithms for operating systems unsuitable for this job.

When a resource request is granted, the resource manager provides a *mapping* specifying the amount and location of the resources allocated to that application. If no feasible mapping can be found, the application will be rejected. Figure 1.3 shows this process, which we define as *run-time mapping*. A resource manager, that takes the resource demand specification of an application and the resource availability of a platform state as input, and produces a resource allocation.

The procedure just described is commonly performed at design-time using semi-automatic tools, that are often still being researched. In this thesis, we even go one step further and perform resource allocation at run-time. Aside from the necessity of run-time resource management in case not all the variables are known to allocate the right set of resources at design-time, additional benefits of run-time resource management can be identified:

- the ability to circumvent hardware faults (*fault tolerance*),
- minimization of operational costs (*energy efficiency*),
- adaptation to user demands (*quality of service*),
- flexibility in the application set (*use-case flexibility*)

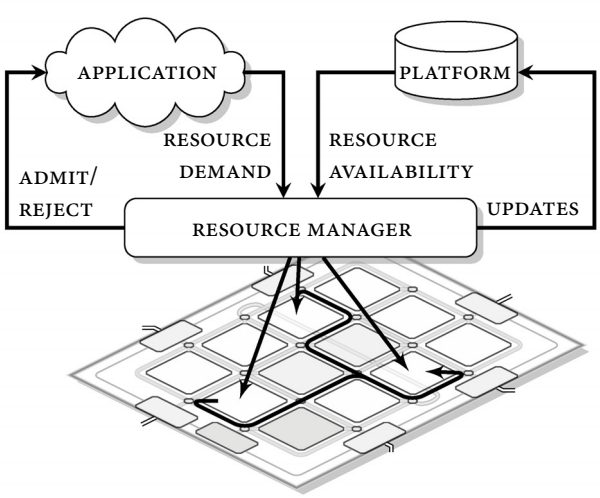


Figure 1.3: Run-time mapping deals with an unknown set of applications and a dynamic platform.

It is the combination of all these features that is most interesting. The aim of this thesis, therefore, is to find a system that provides each of these benefits, at least to some degree.

1.3 The thesis

Given the trend towards heterogeneous computing for energy-efficiency, we see that not only energy for processing but also energy for communication needs to be considered. This trend also has impact on the programmability and resource allocation of heterogeneous many-core systems. Moreover, many desired features of these computing platforms can be achieved by postponing resource management decisions from design-time to run-time. In this way, the obtained flexibility can be exploited to increase the degree of fault tolerance, quality of service, energy efficiency and to support a higher variability in application structure and use-cases, compared to the conventional design-time approach of embedded systems. This work adopts the reservation-based resource partitioning methodology as the abstraction layer between applications and the underlying hardware platform. A main challenge is the complexity of the resource allocation problem, which is also known as the *run-time mapping* problem.

The main hypothesis that this dissertation supports is:

> *Run-time mapping of streaming applications onto large-scale heterogeneous embedded systems is feasible and gives improved flexibility compared to design-time mapping.*

1.3.1 Limiting the scope

Run-time mapping entails many aspects of both embedded systems as well as design methodologies, programming models and performance analysis techniques. In order to limit the scope of this dissertation, the following assumptions and restrictions have been made:

Large-scale distributed memory architectures are commonly used to support high data rates in combination with energy-efficient processing. These features come at a cost of a potentially more complex architecture, compared to symmetric multiprocessing (SMP) platforms. Here, complexity means that the operating system or middleware needs additional control mechanisms to operate the platform, and may need to take additional operational constraints into account. Data locality is key due to efficiency reasons. Techniques such as presented in this thesis are required to control the flexibility that is present in these architectures, and our techniques may be less applicable to architectures not exhibiting distributed memory. [Section 1.1.1]

Streaming applications are timing-sensitive applications that operate on streams of data. The timing constraints on these applications commonly result in fixed platform configurations, derived at design-time. The techniques in this thesis attempt to regain some flexibility in this class of applications, while providing a predictable execution environment to applications in order to sustain their required performance. [Section 1.4]

The message passing programming paradigm is a natural fit for streaming applications. Applications are partitioned in smaller well-defined processes that exchange data through explicit communication. This partitioning allows to reason about the individual pieces of an application, which we define as tasks and channels. [Section 1.2]

Design-time performance analysis is required to derive the amount and type of resources required by an application, without the need to know the exact mapping of the application onto the hardware. For streaming applications, dataflow analysis techniques are available to derive the required resource budgets as a function of the required application performance [TDtB:1].

Online optimization deals with optimization problems that have no or partial knowlegde of the future. In the resource allocation procedure, we consider a single application (request) at a time. Multiple applications (requests) are considered as a stream of events. A resource manager may consider subsequent events in its allocation procedure, but without knowing if and when the next events occur, and what resource demand they have.

1.3.2 Approach

The run-time mapping problem is a very challenging global scheduling problem. A classification of global scheduling techniques is provided by [73], together with their main characteristics and structure. The global scheduling techniques are split

into the classes of deterministic and randomized techniques. *Deterministic techniques* depend on the instance of the problem to be solved and on the effectiveness of the technique for that type of problems [113]. Specific characteristics of the problem are exploited to improve the performance of the technique. These techniques tend to be suitable for run-time application, as they typically only consider a part of the problem's search space. *Randomized techniques*, such as simulated annealing and genetic algorithms, use randomness in the optimization process. They often require a significant amount of computation to reach a solution, which is available at design-time. These techniques start from an initial state and use non-deterministic factors to determine the next state, which is potentially closer to the optimal solution. Randomized techniques typically perform well in solution spaces with several local optima. A fast convergence to a good solution is to be expected in the initial steps of the procedure, but they are less capable of identifying the global (near-)optimal solution to the problem.

In this thesis, both a deterministic heuristic as well as a randomized optimization algorithm are designed and evaluated, in order to solve the mapping problem at run-time. These algorithms must run in an environment with limited resource capacities; that is, little computation time and a few megabytes of memory. The quality of the resource allocation is evaluated in relation to the amount of resources required to execute these algorithms at run-time.

Contributions This thesis is a continuation of [55]. The main contributions of this thesis are the following:

- An integer linear programming formulation of the run-time mapping problem, together with proofs on its computational complexity [TDtB:7].
- A deterministic, domain-specific heuristic that is very fast at the cost of robustness [TDtB:2, TDtB: 3, TDtB: 5].
- A metaheuristical optimization algorithm that is competitive in speed with integer linear programming solvers, but with a memory footprint that is acceptable for embedded systems [TDtB:7].
- A proof-of-concept run-time mapping system as part of the final demonstrators of the CRISP and STARS projects [TDtB:4, TDtB: 5, TDtB: 6].
- A visualization technique that provides insight in the resource availability and usage of a run-time mapping platform.

1.4 Research projects

Both industry and academia actively research technological solutions and programming models that may be applied to a wide range of heterogeneous systems, and hold for more than a few hardware generations [104]. Run-time reconfigurable systems is a key topic of interest in many research agendas. A large part of the results presented in this thesis is obtained in the context of two consecutive research projects that will be introduced next.

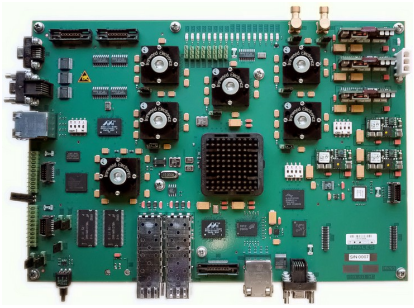


Figure 1.4: The CRISP hardware verification board.

1.4.1 THE CRISP PROJECT

In the sixth framework programme (FP6) project 'Smart Chips for Smart Surroundings (4S)', reconfigurable computing promised to deliver a combination of high performance with energy efficiency and flexibility [99]. The results of 4S revealed new research topics concerning scalability of multi-core systems and dependability of deep sub-micron technologies. On this background, three 4S project partners Atmel Automotive, Recore Systems, and University of Twente joined with NXP Semiconductors, Thales Netherlands, and Tampere University of Technology to form a consortium to break new grounds in scalable and dependable high-performance computing using dynamically reconfigurable many-core platforms.

The 'Cutting edge Reconfigurable ICs for Stream Processing (CRISP)' project investigated a scalable and dependable reconfigurable multi-core system concept that can be used for a wide range of streaming applications in the consumer, automotive, medical and defense markets [TDtB:6]. As a result, a reconfigurable fabric device (RFD) was manufactured, containing 9 Xentium® DSP cores [92]. For verification purposes, five of these RFDs were placed on a board (Figure 1.4), together with an ARM system-on-chip (SoC) and a large field programmable gate array (FPGA). In this research, the concept of run-time mapping was investigated and evaluated on this heterogeneous architecture with 45 DSPs. Chapter 3 describes a greedy, domain-specific run-time mapping algorithm developed during the CRISP project. The performance of this algorithm for various applications, and the concepts enabled by the run-time mapping approach, are described in Chapter 4.

1.4.2 THE STARS PROJECT

The Dutch government funded the 'Sensor Technology Applied in Reconfigurable Systems (STARS)' project to cover six research themes. Two themes are defined around the research topic of this thesis, while most of the other themes are related. Various research tasks within these two themes build upon the research results of the CRISP project. The objective of the STARS project is to develop the necessary knowledge and technology that can be used as a baseline for the development of

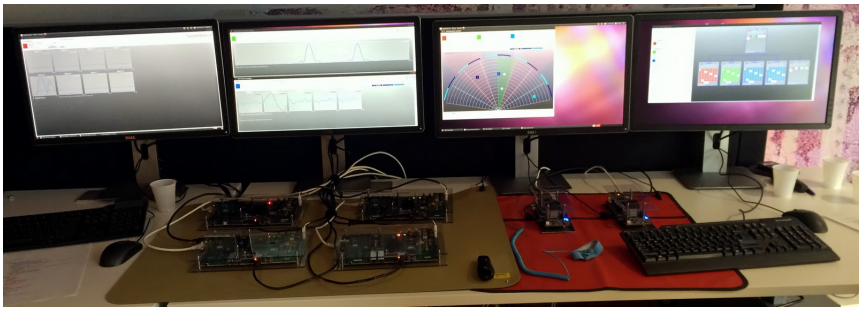


Figure 1.5: The STARS demonstrator setup.

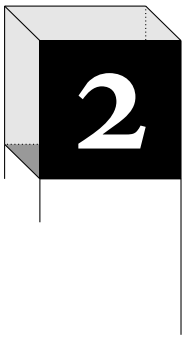
reconfigurable sensors and sensor networks applied in the context of the security domain. Sustainable security systems integrate many functions and need to be highly agile and adaptable to changing circumstances and user requirements. The approach of STARS is to create a versatile, cost-efficient, composable, run-time adaptable system capable of dynamically handling multiple concurrent streaming applications.

Run-time reconfiguration uses information from a design-time synthesis process to find a feasible scheduling of tasks onto the reconfigurable platforms. Newly added applications should then be composable in the sense that they do not disturb the functionality and performance of already operational applications. The allowed time for run-time mapping ranges from the order of seconds until tens of microseconds, depending the hardware platform and application requirements.

Chapter 6 of this thesis describes a meta-heuristic approach to handle the increased scale of the platforms defined in the STARS project. A demonstrator platform has been built, using hardware from the CRISP project. Experiments with this platform are reported in Chapter 5.

1.5 Outline

In Chapter 2, we study a mathematical formulation of the problem central to this thesis, defined as the multi-resource quadratic assignment and routing problem. Together with complexity proofs, some extensions on this problem are given. A deterministic domain-specific heuristic is then presented in Chapter 3. Chapter 4 evaluates the domain-specific heuristic on a multi-processor system-on-Chip (MP-SoC) developed in the CRISP project. Chapter 5 applies the heuristic to a larger platform used in the STARS project. A meta-heuristic for the run-time mapping problem is described in Chapter 6 in order to evaluate the randomized approach to global scheduling problems. Conclusions and recommendations are given in Chapter 7.



2 Mathematical Problem Formulation

Abstract – *This chapter provides a formulation of the run-time mapping problem, which consists of task assignment and communication routing. In mathematical terms, the task assignment is known as the multi-resource generalized assignment problem (MRGAP), and the channel routing is known as the (extended) unsplittable flow problem (UFP), the bandwidth packing problem (BPP), or as the shortest capacitated path problem (SCPP). The combined problem is defined as the multi-resource quadratic assignment and routing problem (MRQARP). An integer linear programming formulation for this problem is provided, as well as complexity proofs on the $\mathcal{NP}$-hardness of the problem.*

The run-time mapping problem roughly consists of two related sub-problems: task assignment and communication (channel) routing. Each of those sub-problems specifies a demand for resources from the underlying platform, which only provides a limited amount of resources. Many practical capacity related problems are variants of the generalized assignment problem (GAP) or bin packing problems. Task allocation in computer systems was already modeled in 1996 as a multidimensional vector packing problem [12]. Since then, many extensions and variations of the problems have been applied to computer systems. An overview of the GAP and its variations is found in [94].

The next section gradually builds up to a integer linear program (ILP) that describes the run-time mapping problem. This ILP is named formally as the multi-resource quadratic assignment and routing problem (MRQARP). As stated in Chapter 1, we focus on a single application and a single platform at at time. Mapping multiple applications to the same platform can be tackled by generating a sequence of problems. These problems are solved iteratively by taking the platform state as defined by the composition of all previously calculated solutions. Extensions on this problem that do consider application sets are described in Section 2.2.

The integer linear programming formulation presented in this chapter is borrowed by Hölzenspies [55] and published in [TDtB:7].

2.1 The Multi-Resource Quadratic Assignment and Routing Problem

Let an application A be specified by a weakly connected and directed multigraph $A = \langle T, C \rangle$, composed of tasks $t \in T$ and channels between tasks $\langle s, d, n \rangle \in C$, with $\{s, d\} \in T$ and index n to differentiate among multiple channels between a pair of tasks, i.e. $C \subseteq T \times T \times \mathbb{N}$. A hardware platform can be described as a directed multigraph $P = \langle E, L \rangle$ with hardware elements $e \in E$ and links between elements $\langle u, v, m \rangle \in L$, with $\{u, v\} \in E$ and m to identify links between pairs of elements, i.e. $L \subseteq E \times E \times \mathbb{N}$. The indices n and m in the channels and links, respectively, can be omitted in cases where the notion of a multigraph is not required, assuming at most one link between a pair of tasks (in each direction). Links can be chained to compose multi-hop paths through a network. Resource allocation for an application then involves the assignment of tasks to elements, and the assignment of the channels between tasks to parts of the interconnect defined by the links between the elements where the producer and consumer tasks of the channel have been assigned. Therefore, we introduce two sets of binary decision variables:

$$x_{te}\text{: specifies assignment of task } t \text{ to element } e$$
$$y_{\langle s,d,n \rangle \langle u,v,m \rangle}\text{: specifies assignment that channel } \langle s, d, n \rangle \text{ uses link } \langle u, v, m \rangle$$

2.1.1 Task to Processor Assignment

The tasks of an application need resources on (processing) elements to be able to execute their functionality. As a single number may not suffice to specify the resource needs, we generalize the problem by modeling resource demands with vectors.

More precisely, the resource need of a task t if is is assigned to element e is specified by a vector r_{te}, which contains a component for every resource type $k \in R$, i.e. r_{te}^k denotes the k^{th} component of vector r, where R composes the set of all distinct resources types [1]. As a dual, the resource capacity vector c_e^k gives the total availability per resource k at element e. Figure 2.1 presents the task assignment subproblem in a network-related formulation, a so-called *netform* [39]. The tasks are shown on the left, whereas the elements are shown on the right. An edge between a task and an element represents a potential assignment. The task assignment subproblem is defined as to select a set of edges, such that each task is assignment to a single element. In doing that, the sum of resources r_{te}^k requested on all selected incoming edges to an element should not exceed its capacity c_e^k.

Multiple resources per task

In the multi-resource generalized assignment problem (MRGAP), a task may require up to $|R|$ different resources from a single (processing) element. This corresponds with a platform containing relatively complex hardware elements, that

[1] Examples of resource vectors and their composition are provided in [55].

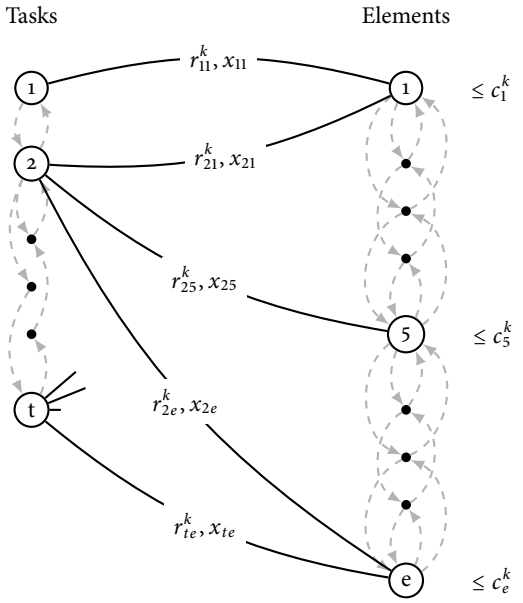
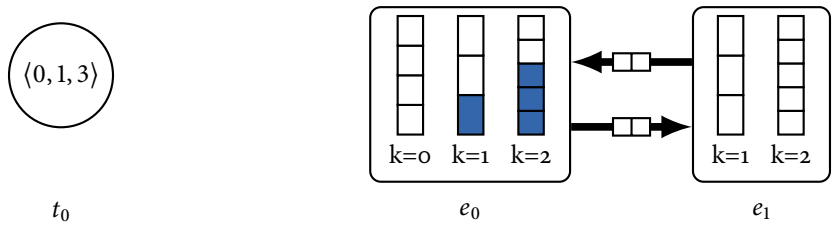


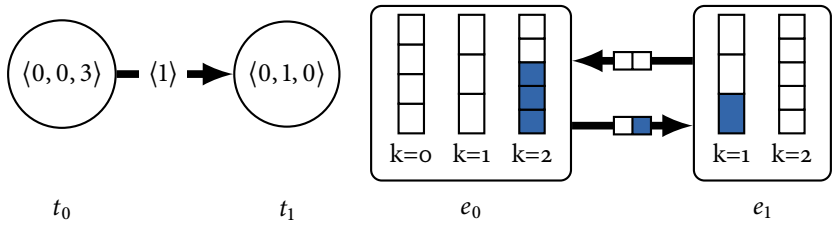
Figure 2.1: The task assignment problem represented in a netform.

embed a number of *tightly coupled* resources. A task may need, for example, a minimal amount of memory within a device to be able to execute its functionality and thus at the same time need computational resources of the same device. These two resources cannot be split and mapped arbitrarily to some location in the platform. An example is given in Figure 2.2a. For this example, we consider three resources ($|R| = 3$) and a task t_0 which needs two of these resources on a single device, as defined by the resource vector $r_{t_0 e_0} = \langle 0, 1, 3 \rangle$. Since the resources are tightly coupled, resource requests may be rejected even if the total remaining capacity of the platform is still sufficient. When the resources requested would be *loosely coupled*, multiple hardware elements may be employed to provide resources. In this case, more solutions may be available compared to the previous case due to a less constrained resource request. An example is given in Figure 2.2b; the request is specified with the vectors $r_{t_0 e_0} = \langle 0, 0, 3 \rangle$ and $r_{t_1 e_1} = \langle 0, 1, 0 \rangle$.

In a GAP formulation, resources are specified as scalars as opposed to vectors in a an MRGAP formulation. Figure 2.2b corresponds to a GAP, which is generalized by the multi-resource quadratic assignment problem (MRQAP) corresponding to Figure 2.2a. In some cases, a GAP formulation suffices, whereas in other cases, an MRGAP formulation is required to express the resource demand.



(a) Tightly coupled resources (MRGAP).



(b) Loosely coupled resources (GAP).

Figure 2.2: Various degrees of resource coupling.

Objective

Next to the hard constraints on the availability of the required resources, different assignments may have different preferences. This preference is modeled in the objective, which is represented in a cost function. For example, such a cost function might differentiate between compatible instruction-set architectures based on the availability of a hardware floating point unit or a varying amount of data or instruction cache, between compatible generations of a protocol standard for I/O interfaces, or between various types of storage (memory, disk). Lower cost means a higher preference for a specific task to resource mapping, indicating a higher efficiency in terms of a performance versus energy ratio. The cost function

$$cost(t, e) \rightarrow \mathbb{Z}^+ \tag{2.1}$$

is specified in such a way that assignments of disallowed pairs of tasks and elements lead to infinite costs.

The costs $cost(t, e)$ are application specific, but there may also be system specific costs expressing (long-term) system-wide objectives. As an example, [43] demonstrates that for an energy minimization objective, the power consumption of the supporting infrastructure has to be considered next to the power consumption of the processing core itself. Other system objectives may include load-balancing and wear-leveling. This means that the cost of mapping a task to a hardware element may also depend on the current configuration or the overall objectives defined by the system. The problem is that individual objectives may be in conflict, requiring a trade-off between application objectives and system objectives such as shown

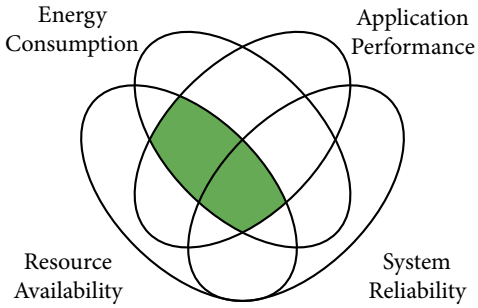


Figure 2.3: Trade-offs in run-time resource allocation between energy consumption, application performance, availability of resources and reliability of the system.

in Figure 2.3. Typical cost functions consist of multiple objectives and consider the amount of resources required. Hence, we define the system specific costs with $cost_{sys}$ and the application specific costs with $cost_{app}$. As the relation between these two cost factors may be unknown, a proportional relationship is established using multiplication. The multiplication makes it easier to compare the two quantities as ratios across all combinations of tasks and elements in the system. For general applicability, the following function composition is assumed to capture the optimization objective:

$$\text{cost}(t,e) = \text{cost}_{app}(t,e) \times \text{cost}_{sys}(e) \tag{2.2}$$

The task assignment problem is known as the multi-resource generalized assignment problem (MRGAP) [36], and is defined as follows:

$$\min \quad \sum_{t\in T}\sum_{e\in E}(cost(t,e)x_{te}), \tag{2.3}$$

$$\text{s.t.} \quad \sum_{e\in E} x_{te} = 1, \quad \forall t \in T \tag{2.4}$$

$$\sum_{t\in T}(r^k_{te}x_{te}) \le c^k_e, \quad \forall e \in E, \forall k \in R \tag{2.5}$$

$$x_{te} \in \{0,1\}, \quad t \in T, e \in E. \tag{2.6}$$

Constraint 2.4 specifies that every task $t \in T$ needs to be assignment to exactly one element $e \in E$; the assignment chosen is stored in x_{te}. The total resource demand r^k_{te} over all tasks $t \in T$ assigned to a specific element $e \in E$ should not exceed its capacity c^k_e, specified by constraint 2.5.

Multiple implementations per task

An extension on the GAP is known as the multi-level generalized assignment problem [95]. This allows a task to be performed at various levels of efficiency. Each

level may specify a different resource demand and associated cost. Next to another level of coefficients, the main difference in the problem definition above would be observed in constraint 2.4. Now precisely one implementation $i \in I$ of a task $t \in T$ is to be mapped to precisely one hardware element $e \in E$:

$$\sum_{e \in E} \sum_{i \in I} x_{tie} = 1, \quad \forall t \in T \tag{2.7}$$

In this work, a task may have multiple implementations, but not more than one for each type of hardware component. This is a restriction we adopt mainly for practical purposes, which can be lifted at the cost of additional computational complexity. A resource demand r_{te} may be specified for each supported combination of task t and element e. Allowing a task to run at various efficiency levels on a specific element e requires the specification of multiple resource demand vectors r_{tie}, one for each implementation $i \in I$.

2.1.2 Communication Routing

A producer task communicates with a consumer task through a channel. The platform must thus provide a communication path with sufficient capacity between these two tasks. If the traffic on a channel can be split over multiple routes, we have the (minimum-cost) multi-commodity flow problem, which can be efficiently solved using a linear programming approach [41]. This splitting may hold for packet-switched networks, where either the network itself performs the routing, or where advanced configuration and control mechanisms that split and join the traffic flows (taking care of the ordering within datastreams). The unsplittable flow problem (UFP)[2] adds the restriction that each communication channel must be routed over a single path through the interconnect. Most work on this problem makes use of the *no-bottleneck assumption*, denoting that all link capacities are larger than the maximum demand that is requested. This guarantees that there is no bottleneck in the network; each channel can be routed over every link. Without the no-bottleneck assumption, the problem is known as the extended UFP [8] and as the constrained shortest path problem [133]. In this thesis, we consider interconnects of embedded systems which typically use a single route per channel to avoid the complexity of reordering packets and the need for additional buffer capacity. Due to the heterogeneous nature of embedded systems combined with quality-of-service guarantees at the application side, it is not guaranteed that a channel can be routed over every possible link while respecting the limited capacity on the link. Hence, this thesis assumes an unsplittable flow problem without the no-bottleneck assumption.

In the communication routing subproblem, channels are defined as task-to-task connections that have to be routed over a single path. The location of both tasks

[2] Other monikers for this problem, or a variant of the problem are the shortest capacitated path problem (SCPP) and the bandwidth packing problem (BPP) [5]. The difference between the SCPP and the BPP is in the cost function, where in the latter the cost of a path is multiplied by the routing demand [24].

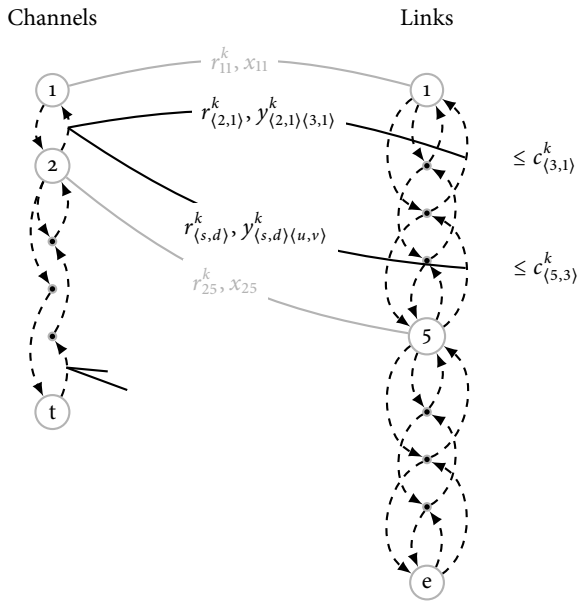


Figure 2.4: The communication routing problem represented in a netform.

is determined by the task assignment subproblem. Figure 2.4 corresponds with Figure 2.1 and presents the routing problem as a netform. In this figure, we assume that task 1 is assigned to element 1 and task 2 is assigned to element 5. Then, each channel between a pair of tasks needs to be assigned to a sequence of links that connect the elements to which the two tasks are assigned. Again, the set of edges (choice) selected in the netform must ensure that the total resource demand per link does not exceed its capacity $c^k_{\langle u,v\rangle}$.

For channel $\langle s, d\rangle \in C$ we have to ensure that the assignment variables y form a path from the element where task s is placed to the element where task d is placed. A communication path is not required in the case that both s and d are assigned to the same element e, i.e. if for all $e \in E$ we have:

$$x_{se} = x_{de} \tag{2.8}$$

In all other cases, the communication path starts at the element to which the producing task has been assigned (we call that the producing element) with one of its outgoing links. So, if the producer task s of channel $\langle s, d\rangle$ is mapped to element u, we require that channel $\langle s, d\rangle$ uses exactly one of the links that leave element u:

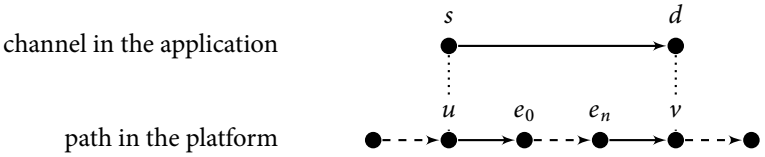
$$\sum_{\langle u,e\rangle \in L} y_{\langle s,d\rangle\langle u,e\rangle} = 1, \qquad \text{if } x_{su} = 1 \tag{2.9}$$

Symmetrically, channel $\langle s, d\rangle$ must use a single incoming link to reach the element v,

to which consumer task d is assigned, i.e. we must have:

$$\sum_{\langle e,v\rangle \in L} y_{\langle s,d\rangle\langle e,v\rangle} = 1, \qquad \text{if } x_{dv} = 1 \tag{2.10}$$

The start segment of the path can be connected to the end segment by chaining a sequence of intermediate links, illustrated as follows:



Therefore, we pose a flow conservation constraint on any element in the network that is not the source or destination of the channel. This constraint ensures that a channel $\langle s, d\rangle$ uses an equal amount of incoming links and outgoing links at any element $e \in E$, for which:

$$\sum_{\langle u,e\rangle \in L} y_{\langle s,d\rangle\langle u,e\rangle} = \sum_{\langle e,v\rangle \in L} y_{\langle s,d\rangle\langle e,v\rangle}, \quad \text{if } x_{se} = x_{de} = 0 \tag{2.11}$$

The various cases (2.8-2.11) concerning the routing problem can be combined into a single constraint (2.12). The links in the platform are furthermore bounded in the amount of communication they can handle. This means that in addition to the logical routing constraints resource capacity constraints have to be added. Constraint 2.13 specifies per link the resource capacity constraint.

$$\text{s.t.} \quad \sum_{\langle u,e\rangle \in L} y_{\langle s,d\rangle\langle u,e\rangle} + x_{se} = \sum_{\langle e,v\rangle \in L} y_{\langle s,d\rangle\langle e,v\rangle} + x_{de}, \quad \forall e \in E, \forall \langle s,d\rangle \in C \tag{2.12}$$

$$\sum_{\langle s,d\rangle \in C} (r^k_{\langle s,d\rangle\langle u,v\rangle} y_{\langle s,d\rangle\langle u,v\rangle}) \leq c^k_{\langle u,v\rangle}, \quad \forall \langle u,v\rangle \in L, \forall k \in R \tag{2.13}$$

$$y_{\langle s,d\rangle\langle u,v\rangle} \in \{0,1\}, \langle s,d\rangle \in C, \langle u,v\rangle \in L. \tag{2.14}$$

Objective

The GAP and MRGAP are generalized in form of the generalized quadratic assignment problem (GQAP) and MRQAP by considering a more complex optimization objective. To specify this objective, for each pair of tasks $\langle t,t\rangle' \in T \times T$ a *flow* is introduced ($flow : T \times T \to \mathbb{Z}^+$) and for each pair of elements $\langle e,e\rangle' \in E \times E$ a *distance* is introduced ($dist : E \times E \to \mathbb{Z}^+$). Then, the function

$$cost_{route}(\langle s,d\rangle) = flow(\langle s,d\rangle) \sum_{\langle u,v\rangle \in L} (dist(\langle u,v\rangle) y_{\langle s,d\rangle\langle u,v\rangle}) \tag{2.15}$$

defines the cost of routing a channel between task s and task d using the sequence of links specified by the y-variables. Note that the set $L \subseteq E \times E$ is used in practice; all potential link between elements $\langle e, e' \rangle \in E \times E$ but not in L have infinite distance. Using the cost function from Equation 2.15, the objective function $cost(t, e)$ is extended to factor in the correlation between any two resources in the platform. Whenever an assignment of task t to element e is considered, the total flow-distance relation to its communication peers $\{t' \in T \mid \langle t, t' \rangle \in C \vee \langle t', t \rangle \in C\}$ is calculated with

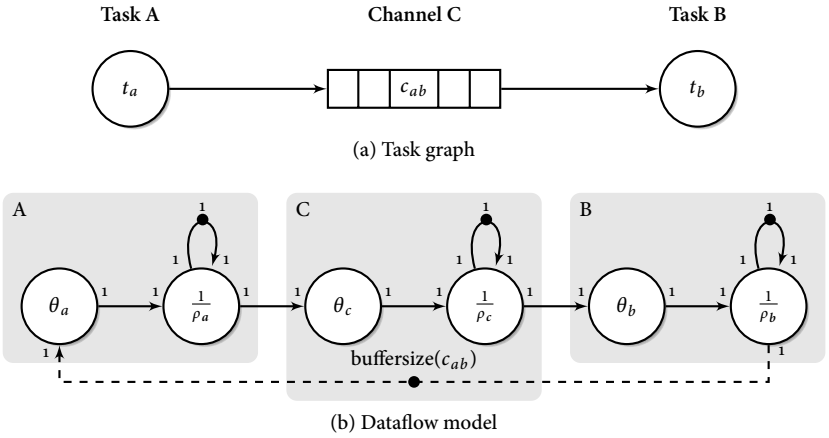
$$cost(t, e) = cost_{app}(t, e) \times cost_{sys}(e) + \sum_{\langle t, t' \rangle \in C} cost_{route}(\langle t, t' \rangle) + \sum_{\langle t', t \rangle \in C} cost_{route}(\langle t', t \rangle). \tag{2.16}$$

This quadratic function is able to express trade-offs in the selection of resources of a specific task in relation to the resources (already) selected for the other tasks in the application graph. As an example, a cost function modeling metrics such as bandwidth (of the communication channels) and power consumption (of the interconnect) ensures that two tasks with a high data exchange are mapped spatially close to each other (assuming that short communication routes take less energy). So, next to the resource constraints of the interconnect (Equation 2.13), the optimization objective differentiates between multiple interconnect routes. Moreover, specific combinations of resources may be preferred over other combinations, independent from the availability of the interconnect. For example, a resource request may opt for a processor in combination with a cluster local scratchpad memory, while the background double data rate (DDR) memory is not an option. The allowed degree of resource coupling is then expressed in the cost function, where disallowed combinations have infinite cost.

2.1.3 Application performance guarantees

The problem formulation presented so far assumes that resources can be split into parts, which in turn are distributable over multiple tasks and / or applications. Memory is a good example of a resource that is easily split up, even in the face of memory fragmentation [93]. The reservation of memory is made in the spatial domain, where specific memory segments are reserved from a larger whole. Once handed out, the resource cannot be (temporarily) taken back without impacting the user; e.g. the data may be overwritten or corrupted.

Other types of resources cannot be split in the spatial domain, but can be shared by having multiple users taking turns in accessing the resource. Using the technique of time slicing, such resources can be shared in the temporal domain. The period during which a user is allowed to use the resource is called a *time slice*; an arbiter ensures that each user is allowed only access to the resource during his time. A schedule is the specific assignment of time slices to users within a given time interval. In order to be able to provide any guarantees on such a schedule, we restrict ourselves to the use of schedulers in the class of *Latency-Rate* ($\mathcal{LR}$) servers [109].



(a) Task graph

(b) Dataflow model

Figure 2.5: Various models capturing different aspects of an application.

A scheduler belongs to the class of $\mathcal{LR}$ servers when a user is allowed access to the resource in every scheduling interval, for at least its reserved time slice:

> *For a scheduling algorithm to belong to the $\mathcal{LR}$ class, it is required that the average rate of service offered by the scheduler to a busy session [user], over every interval starting at time Θ from the beginning of the busy period, is at least equal to its reserved* rate. *The parameter Θ is called the* latency *of the scheduler [109].*

To ensure that tasks complete in a timely fashion, a time budget has to be reserved that corresponds with the minimum rate of performance required. The required time budgets for a given performance level can be derived from performance analysis on dataflow models of the given application [117]. This technique is now explained with an example.

Example 2.1.1 [Performance modeling using dataflow] *Figure 2.5a shows a small application graph having the tasks t_a and t_b communicating via a buffer named c_{ab}. This application graph can be modeled as a dataflow graph. Conventionally, tasks in the application graph have a one-to-one correspondence with actors in the dataflow graph. It is demonstrated in [117] that modeling of run-time scheduling (of resources) is improved by using two actors instead of one per task. The dataflow graph in Figure 2.5b modeling the task graph in Figure 2.5a complies with the assumption of a $\mathcal{LR}$ scheduler. The edges of the graph in Figure 2.5b transfer tokens between actors according to the communication ratio annotated on the start and end of the edges. When sufficient tokens are available on all the input edges of an actor, it is able to execute (to fire). The execution of an actor has a duration which is annotated in the actor node itself, after which it produces the specified number of tokens on its*

output edges. The production of tokens may again enable other actors to execute. Per task in Figure 2.5a, one actor is used to represent the (initial) latency induced on the task when it attempts to access or use the resource in any first scheduling interval, and another actor representing the rate of execution that follows in the subsequent scheduler intervals.

The throughput of an application is a global property; performance analysis has to consider the complete application (graph), including communication between the tasks. Communication can be modeled similar to the way the tasks in Figure 2.5a are represented in Figure 2.5b, consisting of a latency θ component and a rate ρ component. The difference between channels and tasks is that channels may be routed through multiple hops in the network, crossing multiple (hardware) arbiters. The only restriction we impose on the network is that all these arbiters belong to the $\mathcal{LR}$ class. Then, an arbitrary number of $\mathcal{LR}$ servers on a communication path can be modeled by the sum of their individual latencies θ, combined with the minimal rate ρ of the components on the path [109].

Throughput analysis of the dataflow graph makes an assumption on the (maximum) latency of a communication channel. Routing such a communication channel $\langle s, d\rangle \in C$ through an interconnect should ensure that the summation of the individual latencies $\theta_{\langle u,v\rangle}$ of the links $\langle u, v\rangle \in L$ on the route does not exceed the maximum latency $\tilde{\theta}_{\langle s,d\rangle}$, i.e. we have to ensure:

$$\sum_{\langle u,v\rangle \in L} (\theta_{\langle u,v\rangle} y_{\langle s,d\rangle\langle u,v\rangle}) \leq \tilde{\theta}_{\langle s,d\rangle}, \quad \forall \langle s, d\rangle \in C \tag{2.17}$$

In the domain of streaming applications, the performance of an application is commonly expressed in terms of throughput per time unit, e.g. as the number of frames per second an application is required to process. Analysis of the dataflow graph for specific time delays on the actors gives a lower bound on the application throughput [116]. For the example presented in Figure 2.5, Table 2.1 specifies per task a minimally guaranteed time budget in a given scheduling interval. These scheduling settings are modeled in the dataflow graph using the actor attributes as given in the last two columns of Table 2.1. Maximum cycle mean (MCM) analysis [106] of the dataflow graph in Figure 2.5b then gives a cycle mean of

$$\begin{aligned}\mu &= \max(\{\frac{1}{\rho_a}, \frac{1}{\rho_b}, \frac{1}{\rho_c}, \frac{\theta_a + \frac{1}{\rho_a} + \theta_b + \frac{1}{\rho_b} + \theta_c + \frac{1}{\rho_c}}{1}\}) \\ &= \max(\{\frac{4}{2}, \frac{7}{6}, \frac{7}{4}, \frac{2 + \frac{4}{2} + 3 + \frac{7}{6} + 1 + \frac{7}{4}}{1}\}) \\ &= \frac{131}{12}.\end{aligned}$$

The maximum achievable throughput of the dataflow graph, and hence, the application is μ^{-1}, which is $\frac{12}{131} \approx 0.0916$ for the application in Figure 2.5a combined

with the scheduler settings of Table 2.1. If the calculated throughput suffices, the corresponding time budgets can be encoded in the resource vector r^k_{te}. This ensures that a task gets sufficient computation time per interval to maintain a steady throughput.

Table 2.1: Given an application and its schedule, the properties of the corresponding latency-rate dataflow component are obtained.

Application	**Scheduling**		**Dataflow model**	
Task	Budget	Interval	Latency (θ)	Rate (ρ)
t_a	1	3	2	$2/4$
c_{ab}	3	4	1	$6/7$
t_b	2	5	3	$4/7$

2.1.4 Integer Linear Program

In this section, we present the overall formulation of the MRQARP. The used notation is summarized in Table 2.2.

Table 2.2: Notation used to formulate MRQARP.

T	set of tasks in the application;
$C \subseteq T \times T \times \mathbb{N}$	set of channels in the application;
E	set of elements in the platform;
$L \subseteq E \times E \times \mathbb{N}$	set of links in the platform;
R	set of unique resources in the platform;
t	index of tasks $t = 1 \ldots \lvert T \rvert$;
e	index of elements $e = 1 \ldots \lvert E \rvert$;
k	index of resources $r = 1 \ldots \lvert R \rvert$;
$\langle s, d, n \rangle$	channel n between task s and task d, $\langle s, d \rangle \in C$;
$\langle u, v, m \rangle$	link m between element u and element v, $\langle u, v \rangle \in L$;
r^k_{te}	amount of resource k required by element e to perform task t;
$r^k_{\langle s,d,n \rangle}$	amount of resource k required to route channel $\langle s, d, n \rangle$ over a link;
c^k_e	amount of resource k provided by element e;
$c^k_{\langle u,v,m \rangle}$	amount of bandwidth provided by link $\langle u, v, m \rangle$;
$\theta(u, v, m)$	latency of link $\langle u, v, m \rangle$ under the load defined by y;
$\tilde{\theta}(s, d, n)$	the maximum latency allowed for channel $\langle s, d, n \rangle$.

The ILP of the MRQARP is the following:

$$Z = \min \quad \sum_{t \in T} \sum_{e \in E} (cost_{app}(t,e) cost_{sys}(e)\, x_{te}) + \sum_{\langle s,d,n \rangle \in C} (flow(\langle s,d,n \rangle) \sum_{\langle u,v,m \rangle \in L} (dist(\langle u,v,m \rangle) y_{\langle s,d,n \rangle \langle u,v,m \rangle})) \tag{2.18}$$

$$\text{s.t.} \quad \sum_{e \in E} x_{te} = 1, \qquad \forall t \in T \tag{2.19}$$

$$\sum_{\langle u,e,m \rangle \in L} y_{\langle s,d,n \rangle \langle u,e,m \rangle} + x_{se} = \sum_{\langle e,v,m \rangle \in L} y_{\langle s,d,n \rangle \langle e,v,m \rangle} + x_{de}, \qquad \forall \langle s,d,n \rangle \in C, \forall e \in E \tag{2.20}$$

$$\sum_{t \in T} (r^k_{te} x_{te}) \leq c^k_e, \qquad \forall k \in R, \forall e \in E \tag{2.21}$$

$$\sum_{\langle s,d,n \rangle \in C} (r^k_{\langle s,d,n \rangle \langle u,v,m \rangle} y_{\langle s,d,n \rangle \langle u,v,m \rangle}) \leq c^k_{\langle u,v,m \rangle}, \qquad \forall k \in R, \forall \langle u,v,m \rangle \in L \tag{2.22}$$

$$\sum_{\langle u,v,m \rangle \in L} y_{\langle s,d,n \rangle \langle u,v,m \rangle} \theta(u,v,m) \leq \bar{\theta}(s,d,n), \qquad \forall \langle s,d,n \rangle \in C \tag{2.23}$$

$$x_{te} \in \{0,1\} \qquad \forall t \in T, \forall e \in E \tag{2.24}$$

$$y_{\langle s,d,n \rangle \langle u,v,m \rangle} \in \{0,1\} \qquad \forall \langle s,d,n \rangle \in C, \forall \langle u,v,m \rangle \in L \tag{2.25}$$

In the formulation, the objective function (2.18) minimizes the cost of processing the tasks on the elements and the cost of a corresponding routing of channels between the tasks. Equation (2.19) demands that each task $t \in T$ is mapped to precisely one element $e \in E$. Each element e has a capacity vector c_e of dimension $|R|$ to indicate the availability of different types of resources at element e. Constraint (2.21) ensures that these capacity limitations are respected.

The flow conservation constraint (2.20) ensures that for each channel between tasks that are not mapped to the same element, a sequence of connecting links is formed through the interconnect, until the element is reached that is assigned to the consuming task of the channel. For each channel, the number of allocated incoming and outgoing links per element should be balanced (either one or zero). This balance is only influenced by the source and sink elements corresponding to the assignment of the tasks of a channel, starting and terminating the sequence of links respectively. Note that constraint (2.20) allows for cycles in communication paths. There is no need to complicate the model with additional constraints, as due to the cost minimization objective, the best path found for channel $\langle s,d,n \rangle$ always will be a simple path. This holds as we restrict our cost function to the domain of natural numbers $\mathbb{N}^+$. The links in the platform are also bounded in the amount of communication they can handle. In addition to the logical routing problem, we disallow oversubscription of a link's capacity by introducing constraint (2.22),

analogue to the resource constraint Equation 2.21 on the elements. The maximum latency allowed for a communication path is constraint by Equation 2.23.

2.1.5 Computational Complexity

This section presents proofs that the task assignment and communication routing subproblems contained in the MRQARP are computationally difficult problems. The difficulty of these subproblems is shown in relation to known problems with an established complexity status. The class $\mathcal{NP}$ is the set of computational problems for which any solution can be verified with a polynomial time algorithm. Every optimization problem has an equivalent decision problem that decides whether a feasible solution with an objective below a certain threshold exists. For the decision versions of the task assignment and communication routing subproblems, a polynomial verification algorithm exists that sums the demand of all resource allocations and compares them with the bounded capacity of the resource supplies; these decision problems are, therefore, in $\mathcal{NP}$. The complexity class $\mathcal{NP}$-complete contains the problems that are, loosely formulated, the hardest in $\mathcal{NP}$. An optimization problem is in $\mathcal{NP}$-hard if it has a corresponding $\mathcal{NP}$-complete decision problem [114]. A problem is strongly $\mathcal{NP}$-complete or strongly $\mathcal{NP}$-hard if the numerical parameters of the problem have no impact on its classification.

A *reduction* from a problem Y to a problem X is an algorithm which transforms instances for problem Y to equivalent instances for problem X. If such a reduction exists, then problem X is at least as hard as Y because problem X can then be used to solve problem Y. In our case, the 3-partition problem, which is known to be strongly $\mathcal{NP}$-complete [34], is the problem that is reduced to our subproblems.

The 3-partition problem is defined by the following:

3-PARTITION PROBLEM

Input: A finite set A of $3m$ elements, a bound $B \in \mathbb{N}$, and a 'size' $s(a) \in \mathbb{N}$ for each $a \in A$, such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$.

Goal: Partition A into m disjoint sets $S_1, S_2, \cdots, S_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in S_i} s(a) = B$.

Firstly, we consider the communication routing subproblem in a reduction of the 3-partition problem to the MRQARP. We construct a problem instance with an application having two tasks, and a system consisting of two elements, with a given one-to-one mapping of the tasks to the elements (see Figure 2.6). The sets $S_1, S_2, \cdots, S_m$ of the 3-partition problem are then represented by m links in the platform between elements e_0 and e_1. The set A is represented using $n = 3m$ channels between task t_0

and t_1. The size $s(a)$ for each $a \in A$ is encoded as the resource demand of the corresponding channel $\langle t_0, t_1, a\rangle \in C$. The disjoint sets $S_1, S_2, \cdots, S_m$ are represented by m links between elements e_0 and e_1, each having a capacity equal to bound B. The goal is then to route all the channels in $\langle t_0, t_1, n\rangle \in C$ over the links in $\langle e_0, e_1, m\rangle \in L$ without oversubscribing the link capacity $c^0_{\langle u,v,m\rangle} = B$. For any valid solution, it holds that

$$\sum_{\langle t_0,t_1,a\rangle \in C} r^0_{\langle t_0,t_1,a\rangle} y_{\langle t_0,t_1,a\rangle\langle e_0,e_1,m\rangle} \leq c^0_{\langle e_0,e_1,m\rangle}, \quad \forall \langle e_0, e_1, m\rangle \in L$$

With this reduction of the 3-partition to the communication subproblem, we have established that the equivalent decision problem of communication routing is $\mathcal{NP}$-complete.

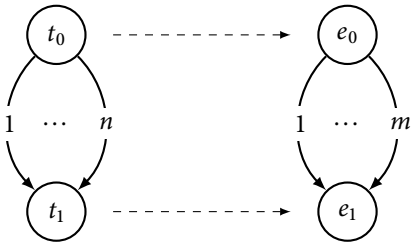


Figure 2.6: Reducing the 3-partition problem to the communication routing problem shows $\mathcal{NP}$-completeness in the strong sense.

Secondly, we consider the task assignment subproblem. For this reduction, we construct an MRQARP instance with $|E| = m$ elements and $|T| = n = 3m$ tasks, where the platform is a complete graph in which the links have infinite capacity. This results in a trivial communication routing problem, and thus only the complexity of assigning the tasks is considered. The set A of the 3-partition problem is represented by the $n = 3m$ tasks. The size $s(a)$ for all $a \in A$ are encoded in the resource demand of the tasks $t \in T$. The disjoint sets $S_1, S_2, \cdots, S_m$ are represented by m elements, each having a capacity equal to bound B. In this reduction, the tasks have to be partitioned in sets of tasks to verify whether enough resources are available on their assignment element x_{te}. Thus, the 3-partition problem can be reduced to the decision problem of the task assignment problem in polynomial time. Hence, the decision version of MRQARP is $\mathcal{NP}$-complete in the strong sense, but even more, this $\mathcal{NP}$-completeness comes back in two different parts of the problem.

Fully polynomial-time approximation schemes do not exist for strongly $\mathcal{NP}$-hard problems[3], unless $\mathcal{P} = \mathcal{NP}$ [35]. Furthermore, based on the previous given reductions, even a decomposition of the problem into a hierarchical 2-step approach still gives an $\mathcal{NP}$-hard optimization problem in the second stage. The problem is made difficult by the limited resource capacities; i.e. by constraints (2.21) and (2.22). Despite the computational complexity of MRQARP, the overdimensioning of practical

[3] Having integral cost and a polynomial bounded objective function.

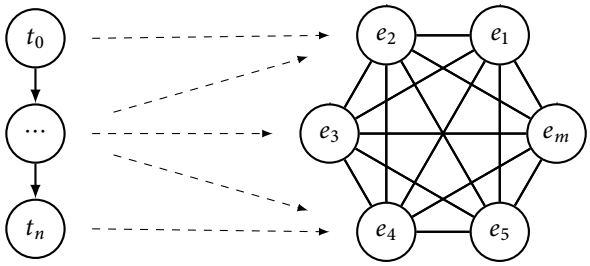


Figure 2.7: Reducing the 3-partition problem to the task assignment problem shows $\mathcal{NP}$-completeness in the strong sense.

systems usually allows us to find feasible solutions in reasonable time, as shown in subsequent chapters.

2.2 Problem Extensions

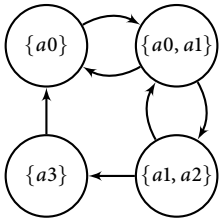
A high quality solution for an individual application may have adverse effects on the consecutive list of applications that may have to be mapped to the same platform. In this section, two extensions to the ILP formulation are provided to reason about multiple applications; we introduce admission control and use-cases. A specific use of the latter is to model time intervals, where in each interval an application is removed or added to the system. To this end, we introduce some additional notation first:

A	set of applications;
T_a	set of tasks in application a;
$T = \bigcup_{a \in A} T_a$	set of all tasks of all applications;
$C_a \subseteq T_a \times T_a$	set of channels between tasks in application a.

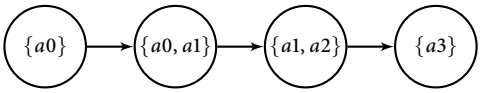
2.2.1 Use cases and time intervals

Most applications in a running system have a limited lifespan. Events that denote the activation and deactivation of applications split the time horizon into multiple intervals. During each interval a specific set of applications is active. Each of such a set of applications is defined as a *use case*. To harness the complexity of system design, a limited number of use cases may be considered, with predefined transitions between them. So, use cases may be useful if the (sets of) applications (ie. (A, T, T_a, C_a)) are known in advance, but their activation time and lifespan are not known on beforehand (Figure 2.8).

A transition from one use case to another indicates a change in the set of active applications. A use case transition may result in multiple activations and/or deactivations in the active application set. Applications that are active both before and



(a) A system with four use cases.



(b) A timeline with four intervals.

Figure 2.8: Problem extensions with discrete time intervals.

after the transition must be considered to be unaware of events happening in other parts of the system, and thus should be left untouched. For these applications, constraints are specified that their resource reservation must remain the same on both sides of the transition. Figure 2.8a shows a system with four use cases, in which a transition is defined from $\{a0, a1\}$ to $\{a1, a2\}$. In this use-case transition, $a0$ is removed and $a2$ is added, whereas $a1$ should be unaware of the events taking place.

The problem formulation of Section 2.1.4 considers a single resource request of a single application. A special set of use cases can be constructed to model a timeline of activation and deactivation events. Each event on the timeline, illustrated by Figure 2.8b, is then restricted to having at most one incoming transition and one outgoing transition. Such a problem instance is used to solve a sequence of problems in a clairvoyant manner; i.e. if the future is known, a solution may be found that improves over a solution that contains suboptimal decisions at previous time steps that cannot be changed.

The following notation is introduced to formulate the use-case extension:

$U \subseteq \mathcal{P}(A)$	set of use cases, as powerset of the application set A,
$Q \subseteq U \times U$	set of use case transitions,
x_{qate}	binary variable assigning task t from, application a in use case q to element e,
$y_{qa\langle s,d\rangle\langle u,v\rangle}$	binary variable assigning channel $\langle s, d\rangle$, from application a in use case q to link $\langle u, v\rangle$.

Use cases add another dimension to the MRQARP, which vastly increases the complexity (scale) of the problem. For this extension, the variables x and y in the problem formulation of Section 2.1.4 have to be augmented with dimension Q. An application a that exists on both sides of a use case transition $\langle q_1, q_2\rangle \in Q$ shall

have the same task mapping and communication routing in both use-cases. These constraints are specified as 2.26 and 2.27:

$$x_{q_1 ate} = x_{q_2 ate}, \quad \forall t \in T_a, \forall e \in E_i, \forall \langle q_1, q_2 \rangle \in Q, \forall a \in q_1 \cap q_2 \tag{2.26}$$

$$y_{q_1 a \langle s,d \rangle \langle u,v \rangle} = y_{q_2 a \langle s,d \rangle \langle u,v \rangle}, \quad \forall \langle s, d \rangle \in C_a, \forall \langle u, v \rangle \in L, \forall \langle q_1, q_2 \rangle \in Q, \forall a \in q_1 \cap q_2 \tag{2.27}$$

2.2.2 OVERSUBSCRIBED SYSTEMS

Admission control is a mechanism to shield a system from oversubscription. The decision on whether or not a single application may be admitted to the platform is obtained by solving the MRQARP. In case not all applications can be admitted to the system, the most beneficial subset of applications should be determined. A common property used is the relative priority between applications, where the applications with the highest priority are preferred over applications with lower priority. However, it is allowed to admit lower priority applications when a higher priority application cannot be admitted to the platform due to resource scarcity. We introduce a function *revenue(a)* over A that gives the revenue for each application. A revenue is only rewarded if an application is indeed mapped onto the platform. A variable $S_a \in \{0,1\}$ indicates whether the application will mapped onto the platform. The revenue is only rewarded if $S_a = 0$. The value of S_a results from whether all tasks of the application are mapped. This is formalized by the following definition:

$$S_a = \begin{cases} 0 & \text{if} \quad \sum_{t \in T_a} \sum_{e \in E} x_{ate} = |T_a| \\ 1 & \text{else if} \quad \exists_{t \in T_a} : \sum_{e \in E} x_{ate} = 0 \end{cases} \tag{2.28}$$

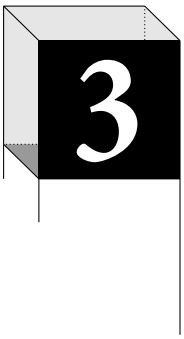
With admission control, each task is assigned to a processing element exactly once, unless the application as a whole is rejected ($S_a = 1$). In the MRQARP formulation, Constraint 2.19 is replaced by Constraint (2.30). Instead of a cost minimization objective, the optimization criteria in oversubscribed systems is changed to maximize the revenue:

$$\max \quad \sum_{a \in A} (\text{revenue}(a) * (1 - S_a) - \text{cost}(a)) \tag{2.29}$$

$$\text{s.t.} \quad \sum_{e \in E} x_{ate} + S_a = 1, \quad \forall a \in A, \forall t \in T_a \tag{2.30}$$

2.3 Conclusions

The run-time mapping problem consists of two optimization problems that are both $\mathcal{NP}$-hard in the strong sense, proven by reduction of the 3 partition problem to these problems. Disregarding any domain-specific knowledge, these properties do not provide any hints on the ordering in which the task assignment and channel routing sub-problems have to be solved. Both sub-problems consider limited capacities of resources, which is the main reason for the inherent complexity. The best matching problem in literature is either the GQAP or MRQAP, of which the latter allows for more complex platform models. The problem extensions to model time intervals, use cases and oversubscribed systems are provided for design-time analysis and design-time synthesis of configurations.



3 Domain-Specific Heuristics: BFS2GAP

Abstract – *Applications and platforms are both modeled as weakly connected graphs of tasks and (processing) elements, respectively. A domain-specific heuristic traverses the graphs simultaneously while assigning tasks to elements, and routing communication channels through the interconnect. Empirical validation of the proposed heuristic shows execution times in the order of 10s of microseconds on a low-end embedded processor for synthetic applications with up to 16 tasks. While having good performance, the heuristic lacks robustness when the approach fails to find a solution. Feedback information on what part of the problem instance causes difficulties is limited.*

Run-time mapping has long been a research topic of the computer architecture for embedded systems (CAES) group. While the importance of the problem is confirmed by the interests and needs of industry, a fundamental question was not answered in the past. Due to the complexity of the problem needing to be solved, the main question is whether the response time and overhead of the run-time mapping mechanism is adequate for application in industrial systems or not. In the attempt to answer this question, we prioritize the reduction of memory and processing overhead over solution quality. In this chapter, first related work is described that influenced the development of the domain-specific heuristic presented second, followed by an empirical validation with a synthetic dataset. The domain-specific knowledge is gained from the CRISP project, in which a fully functional demonstrator is built containing the proposed algorithms. Chapter 4 provides background information on this project together with results of real-world case studies.

The contents of this chapter has similarities to and extends upon Chapters 4 and 5 of the dissertation *On run-time exploitation of concurrency* [55], the result of close cooperation between both authors. The results are also published in [TDtB:1, TDtB: 2, TDtB: 3, TDtB: 6].

3.1 Related work

An extensive survey[1] on mapping methodologies for multi- and many-core systems is presented in [104]. The referenced work is organized in a taxonomy that considers design-time versus run-time mapping, homogeneous versus heterogeneous architectures, and centralized versus distributed management. The focus of this thesis is on run-time mapping for heterogeneous architectures, implemented with either centralized management, or with a combined centralized and distributed management. Among the upcoming trends listed in [104] are *hybrid mapping*, *large scale architectures* and *3D integration of cores*. Design time analysis is leveraged in hybrid mapping (strategies) for run-time use, resulting in less overhead or a more efficient configuration. The upcoming use of large scale architectures requires distributed resource management to cope with scalability issues. The introduction of 3D heterogeneous architectures increases the demand for efficient mapping methodologies to deal with proper thermal management and unreliable system behavior. These trends are considered in the research and development of the mapping techniques presented in this thesis. In the remainder of this section we discuss some related work that specifically inspired the mapping techniques presented in Chapters 3 to 5, organized by topic. We start with a straighforward approach known as *first fit* mapping.

First Fit mapping

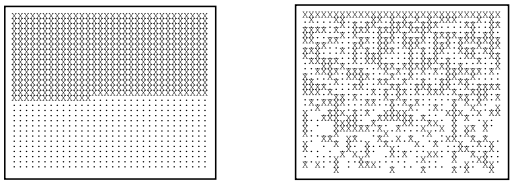
In some works [57, 81], the low-complexity First Fit (FF) algorithm is used for the initial mapping of tasks to processing elements (PEs). After this greedy assignment, Hölzenspies et al. [57] try to improve the quality of the mapping by relocating tasks. Iteratively for each task, an attempt is made to find a better assignment by probing other locally available PEs of the same type. In this operation, it is allowed to swap two tasks that are mapped to the same PE type. When the assignment does not improve, the previous mapping is restored.

Moreira et al. [81] support multi-tasking by mapping tasks to intermediate virtual tiles (VTs). Then, each VT is assigned to a physical PE, taking communication channels into account. The 'first fit with clustering' algorithm of Moreira et al. [81] is more tailored to their architecture than a general first fit algorithm. In their architecture, each router is connected to three PEs, making it desirable to map heavily communicating tasks to the same cluster (of three PEs). Although the clustering algorithm decreases bandwidth usage, it has an adverse effect on the mapping success rate when the system becomes saturated [81]. Therefore, Moreira et al. [81] use an unclustered version of their algorithm when it fails. This point at which the clustering is disabled seems to be related to the degree of fragmentation of the resources.

[1] Part of our work, published as [TDtB:3], is included in the referenced survey.

Fragmentation

Not all systems allow for arbitrary task migration from one core to another. With this assumption that tasks cannot (always) be migrated to other resources, a mapping is fixed for the lifetime of an application. A new resource allocation request, thus, has to deal with the actual configuration state of the system. This may require a mapping algorithm to consider the use of the platform's resources across multiple resource allocation requests. In the illustrative Figure 3.1, two applications are mapped to a platform, using a simple linear allocation method. In each step, the resources required by one of the applications are allocated or de-allocated, while their precise resource requirements are varied over time. As can be seen in the figure, the platform is highly fragmented after 25 time steps. Although the sum of the resources required by the application may be smaller than the available resources in the platform, the routing may fail if the resources are too widely spread over the platform. Consequently, resource fragmentation must thus be taken into account, as future resource allocation attempts are more likely to fail when the fragmentation increases.



(a) Initial state (b) 25_{th} reconfiguration

Figure 3.1: Resource fragmentation of a platform with two applications [69].

Platforms suffer both from internal and external fragmentation of available resources. A task may not employ an element to its fullest, leaving some percentage of the available resources unused. *Internal fragmentation* is the result of allocating a set of resources that does not precisely fit a certain amount of elements. *External fragmentation* results from allocation of non-adjacent PEs, as in Figure 3.1b. In [77], these terms are expressed as ratios to the total amount of resources. In this thesis, external fragmentation of resources is defined in a similar manner.

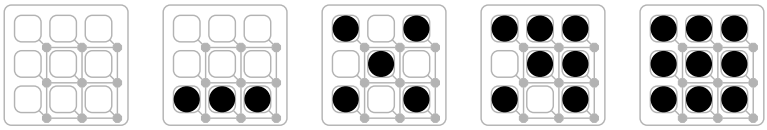
Definition 1 ⌊External fragmentation⌉ *Let α be the number of adjacent element pairs, where the first element is used by some task $t \in T$, but the second is not:*

$$\alpha = |\{\langle u, v\rangle \in L \mid \exists_{t \in T} M(t) = u \wedge \nexists_{t \in T} M(t) = v\}|$$

Then the external fragmentation of a platform P is expressed as a ratio to the total numer of adjacent elements in the system:

$$f_{ext}(P) = \frac{\alpha}{|L|}$$

With this definition, the external fragmentation of a completely empty or fully utilized platform is 0%. The worst external fragmentation possible is a chessboard pattern, illustrated with Figure 3.2c. According to the definition, the external fragmentation of such patterns is 100%.



(a) f_{ext} = 0% (b) f_{ext} = 25% (c) f_{ext} = 100% (d) f_{ext} = 50% (e) f_{ext} = 0%

Figure 3.2: External fragmentation on some example platforms.

In related work, fragmentation is sometimes considered to be a problem, or sometimes dealt with implicitly. For example in [81], algorithms are "adjusted to fill up partially filled PEs" to reduce internal resource fragmentation. The trade-off between fragmentation reduction and other objectives like minimal energy consumption or use of communication resources is not mentioned in [81]. The next section refers to influential work that fully integrates fragmentation reduction in their approach.

Mapping with region selection

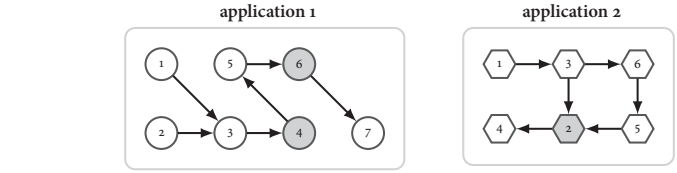
Figure 3.3 shows two alternative mappings of two applications, where after mapping the second application, fragmentation of the available PEs becomes significant (see: Figure 3.3b) when using a greedy mapping algorithm. In the fault-aware resource management approach of Chou and Marculescu [19], a *region selection* procedure is proposed that allows for a more effective mapping for any subsequent applications. The region is obtained by picking PEs around the initially chosen PE that have the lowest total value by the following definitions:

Definition 2 ⌊Dispersion factor⌉ *The* dispersion factor $D(e)$ *of a PE e, is defined as the number of idling neighbors of that PE.*

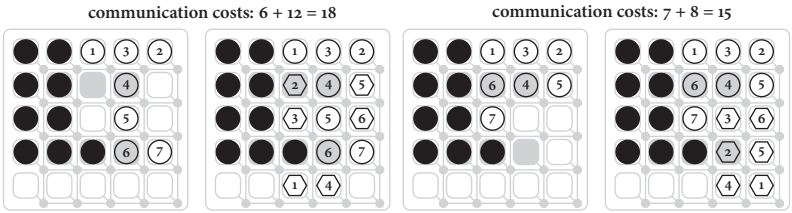
Definition 3 ⌊Centrifugal factor (C)⌉ *The* centrifugal factor $C(e)$ *of a PE e, is defined as the Manhattan distance between PE e and any PE at the border of the region occupied by the current application.*

A PE that has most of its neighbors utilized, *i.e.* it has a small dispersion value, is very likely to be later isolated; selecting this PE to be utilized for the application mapping helps reducing the external fragmentation. A PE with the smaller centrifugal value is better suited for selection, since it increases locality of the tasks in the current application. After a region has been determined by selecting the right number of PEs according to these definitions, the resource manager attempts to find the best mapping within that region. Any mapping algorithm can be used to

allocate PEs within the selection region using some given objective. Chou and Marculescu [20] propose the *node coloring* method, which aims at low inter-processor communication. In the node coloring algorithm, all tasks that need to be mapped are initially colored as white. In a non-increasing order of communication volume, tasks are assigned to processing elements in a two-stage heuristic. In a discovery stage, white tasks are colored grey after candidate PEs are selected. In a finalization stage, a grey task t is mapped if all its neighbor tasks have been colored grey or black. In the mapping decision, the PE is selected with a minimum distance to the PEs of its neighbor tasks. The region selection algorithm combined with the node coloring technique shows a 25% reduction in inter-processor communication compared to the same mapping algorithm without the region selection strategy. Figure 3.3c shows the result of the region selection procedure applied to the example in Figure 3.3a. The region selection approach has cost 15, which is an improvement over the greedy approach with cost 18.



(a) Two applications that have to be mapped



(b) Greedy mapping approach. (c) Mapping approach with region selection.

Figure 3.3: An example of two approaches in incremental application mapping on a heterogeneous MPSoC, where both the degree of fragmentation and communication costs play a role.

The proposal of Chou and Marculescu [20] best suits dense application graphs, while a sparse graph could have a straight line of PEs as a very good solution. Thus, in case of a communication minimization objective it is sufficient (and more general) to minimize the distance between neighboring nodes, like in [57]. Unfortunately, the authors do not evaluate the mapping success rate, which we expect to increase (in the system model used in [20]) when the degree of fragmentation is lowered.

Extending the approach of [20], Chou and Marculescu address fault-aware resource management in [19]. Chou and Marculescu state that faults in an MPSoC result in irregularity, while assuming a 6x6 meshed MPSoC platform containing only two types of tiles; computational cores and memories. Their approach results in highly clustered mappings of applications, which seems to conflict with the potential irregularity of the system's resources. Moreover, in most platforms communication becomes the bottleneck, especially when accessing memories or I/O ports. In the illustrative platform of Figure 3.3c, the I/O components would likely be presented at the border of the platform. A typical application that is processing input in order to provide the results on an output component may cross the entire system, depending on the location of the I/O components. Therefore, task mapping is directly related with communication routing.

Communication routing

An interconnect topology must be exploited effectively; how a network is used is determined by the routing strategy. In macroscopic networking, routing is based on dynamically constructed tables which map addresses to routing hops. For high performance computing this approach is already considered prohibitive, so one should not expect it to be usable for a network-on-chip (NoC) in general. General purpose architectures allow for routes to be constructed on demand. In the domain of embedded systems the situation is different: communication partners rarely change [57]. Rather than depending on addressed routing, often small semi-static source routing tables are used. Routes are computed and adapted at design-time [46, 47] or at run-time [57, 78, 86]. The latter approach is followed in our platform.

Energy consumption is used in [19] as an optimization or decision criterion. Realistic models of the energy (in)directly consumed by applications are too detailed for evaluation at run-time. Simpler models are too inaccurate, due to deviations of the application behavior to the specification and due to the layout of the architecture and variations in silicon. The platform model proposed in this thesis is capable of describing heterogeneous interconnects. In this model, we use a cost factor to distinguish between different types of communications links, and assume that longer communication routes take more energy. Longer communication routes likely come with higher communication latencies; these may have adverse effects on the application performance, if not taken into account.

Dataflow driven mapping

The work of Stuijk et al. in [111] performs a combined mapping and scheduling procedure that depends on design-time analysis of the application using dataflow analysis techniques. The objective of the resource allocation approach is to balance the load of the application equally over all the PEs. The load of a PE is estimated by the relative processing performed, the memory used and the number of connections and associated bandwidth. The mapping algorithm processes the tasks

in order of the criticality of the cycles in the corresponding dataflow graph that contain the task at hand. For each task, the PE with the least load is selected by considering the partial mapping during the execution of the mapping algorithm. When no PE is found for a given task, the problem is considered infeasible; no backtracking mechanism is used. Stuijk et al. use dataflow analysis techniques to specify the cost functions used to order the tasks and PEs. The actual mapping algorithm is somewhat similar to the algorithms presented in [85]. Both approaches embed most of the domain knowledge into the cost function, rather than in the mapping algorithm itself. These cost functions may become quite complex and expensive to evaluate, especially if multiple objectives are factored in.

Pareto-based optimization

Multi-objective optimization is the process of simultaneously optimizing two or more conflicting objectives, subject to constraints. The best solutions for multi-objective problems are called the Pareto set. For solutions from the Pareto set, it is not possible to select another solution to improve one of the objectives, without worsening at least one of the others [108]. Ykman-Couvreur et al. [127, 128] use design-time exploration to construct a Pareto set that contains multiple implementations of an application. In a similar way, design-time exploration can produce a Pareto set for each task within an application.

In the work of Shojaei et al., Pareto-optimal application configurations are determined at design-time. Driven by activation and deactivation events of applications, the problem solved is to find the best combination of configurations. To this end, Shojaei et al. present a heuristic to solve multi-dimensional multiple-choice knapsack problems (MMKPs) [101, 102]. While the problem allows for multiple dimensions in the modeling of the resources, it seems that only task assignment is considered.

In [127], an MMKP formulation is used to obtain exact or near-optimal solutions for allocation of resources. Ykman-Couvreur et al. [127] developed a heuristic for solving the MMKP. Their heuristic reduces the multi-dimension resource requirements of each item into a single resource combination with a certain cost, ordered by *value per resource combination*. The last step is a greedy algorithm that optimizes the total value of the chosen resource combinations until there is no better feasible solution available. This greedy selection can be applied to any sorted optimization points. While this approach is derived from a mathematical formulation of the optimal solution, simpler sorting algorithms may lead to similar results. Other works use energy consumption [31, 57] or execution time [83] as an ordering criteria. Faruque et al. [31] specify the energy consumption of a PE type with two values, namely static and dynamic energy. Hölzenspies et al. [56] differentiate in energy consumption per architectural type, normalized by their processing capacity.

Distributed management

Reasoning about the global state of the platform often results in centralized approaches [84]. For larger platforms, centralized resource management might not scale enough. In [103], a central entity initially distributes applications into virtual clusters, whereafter a packing strategy tries to map their tasks close to each other. In [134], a decentralized heuristic is presented that combines a spring layout approach with Tabu search. A combined distributed and fault-tolerant approach is presented in [58], by integrating the task mapping algorithm into the routing protocol.

3.1.1 Taxonomy

Finding an optimal solution to add applications to an MPSoC is very complex. To avoid this complexity, most researchers split the problem into multiple sub-problems. In this section, we present a taxonomy for these sub-problems. For brevity reasons, a specific ordering is assumed, but different orderings exist as well. Each sub-problem is attacked iteratively in a separate phase, with reduced complexity compared to the original problem. Hölzenspies et al. [57] name such a work-flow *hierarchical search with iterative refinement*. The steps required for resource allocation can be classified into four phases:

1. **Binding:** for each task of the application, an implementation is selected that is able to execute the task with low cost and sufficient high performance. The resource requirements of the chosen implementations, including the type of architecture for execution, must be satisfied to admit the application.
2. **Mapping:** taking locality into account, specific resources are assigned to each task, such that the resource requirements of the implementation chosen in phase 1, are fulfilled.
3. **Routing:** for pairs of tasks that need to communicate, communication links are established between the processing elements assigned to them in phase 2. The amount of communication resources required is taken into account.
4. **Validation:** the performance constraints that are specified in the application specification are validated against the performance that is guaranteed by the resource allocation derived in the previous phases.

Figure 3.4 illustrates this taxonomy. With the 'partitioning' phase, we indicate the software development effort at design-time. We only consider the *binding*, *mapping*, *routing* and *validation* phase to be part of a run-time resource manager. As a result of these phases, a resource allocation is provided. Platform specific software can configure the hardware accordingly and start the application, which we indicate with the 'initialization' phase.

The next section presents a mapping heuristic that is inspired by the related work discussed in this section. The key concepts are a form of dynamic region selection, exploitation of communication and interconnect topologies, an unbounded number of resource types and considering both fragmentation and communication

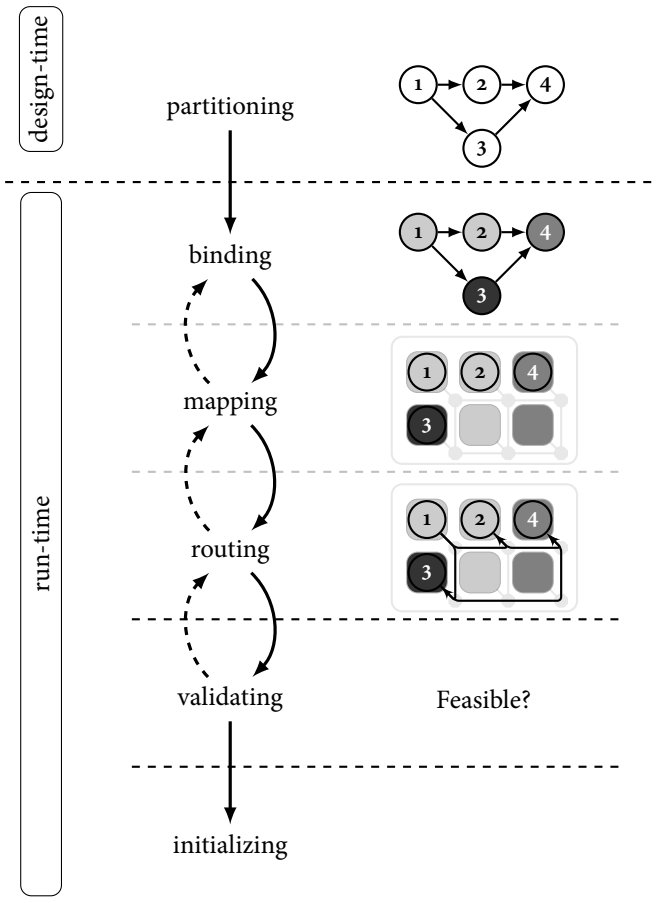


Figure 3.4: System-level configuration consisting of multiple phases.

minimization. The aim of the presented heuristic is to deal with architectures with a high degree of heterogeneity and irregular topologies in order to better match the next generation of platforms.

3.2 A DOMAIN-SPECIFIC MAPPING HEURISTIC

Difficult problems may require a lot of computational effort to find a solution. A *heuristic* is a (set of) rules that use domain-specific knowledge to decrease the computational effort required to find a solution. This is commonly achieved by trading optimality of the solution for feasibility.

The domain-specific structures to be exploited in the heuristic presented in this section are found in the application graph. More precisely, we focus on the property

that applications require a minimal amount of I/O activity to provide meaningful functionality (we consider applications without any input or output tasks not to be real world applications[2]). The main assumption underlying this chapter is that at least one (I/O) task within an application has a fixed mapping in the platform. While I/O operations may be generic in nature, specific locations to retrieve data from and to store results must be fixed, before the application is actually started. For example, an application may be developed such that it can take its input data from any Ethernet port; upon starting the application the user is typically expected to specify the specific port to use for that instance.

Let $T_0 \subseteq T$ be the subset of tasks in application A that can be mapped to only a single element (with the assumption of at least one fixed I/O task, this set is always non-empty). Substantiating these relations results in an injective and non-surjective mapping $M_0 : T_0 \to E_0$, where $E_0 \subseteq E$. The initial set T_0 of tasks is used to divide the allocation problem into subproblems, indexed by i. Each sub-problem i is then defined as a subset of tasks $T_i \subseteq T$, such that T_i is the i^{th} undirected neighborhood N_i of T_0. In the context of graphs, the neighborhood $N(v)$ of a vertex (v) is defined as follows:

Definition 4 ⌊Neighborhood⌉ *In a graph, the out-neighborhood $N^+(v)$ and in-neighborhood $N^-(v)$ of vertex v define the direct successors and predecessors of v, respectively. The neighborhood $N(v)$ of vertex v is then equal to the union of the out- and in-neighborhood. Vertex v itself is never a member of the set $N(v)$. The i^{th} neighborhood of v, written as $N_i(v)$ contains the elements that lie on a distance i of v, defining distance as the number of edges on the shortest path through the graph.*

The proposed heuristic employs a divide-and-conquer approach [76], consisting of the following three steps:

1. Group the tasks in sets with equal distance to the origin task(s) $t \in T_0$.

Maintaining the order of increasing distance, each sub-problem i is then resolved in an iterative manner by these two steps:

2. Search the platform for enough elements $E_i \subseteq E$ spatially close to E_{i-1}, such that the resource requirements of the tasks in T_i can be met.
3. Find a mapping M_i of the tasks in T_i to the elements in $E_i \subseteq E$.

Step 2 introduces a dependency between the current iteration i and the previous iteration $i-1$ of step 2, by considering spatial locality of tasks in the mapping. The tasks in the application are thus mapped in an iterative manner.

Definition 5 ⌊Iterative mapping⌉ *Let $T_i \in T$ be a set of tasks in iteration i, and $E_i \subseteq E$ likewise. Then M_i is a mapping from T_i to E_i in iteration i, such that $\forall\, i \neq j \mid M_i \cap M_j = \emptyset$. For convenience, we define $M_i^* = \bigcup_{\forall j \leq i} M_j$.*

[2] Applications that violate our assumption are supported though.

This approach is illustrated in Figure 3.5, and will be the subject of the following two sections. Section 3.2.1 describes the problem decomposition (divide), while Section 3.2.2 shows how the sub-problems are solved (conquer). Altogether, these three steps make up the heuristic presented in this chapter.

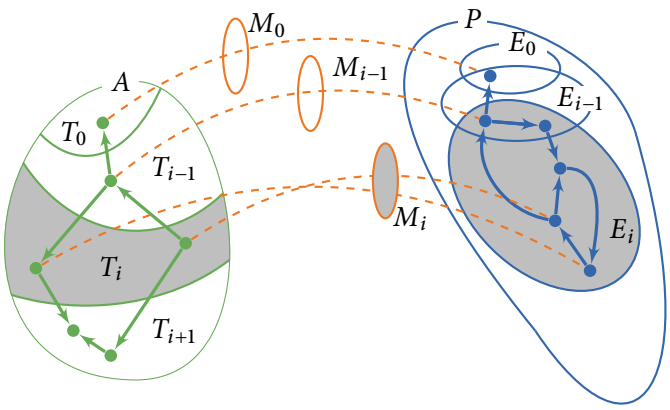


Figure 3.5: The resource allocation problem is partitioned into multiple sub-problems, using the structure of the task graph.

3.2.1 SEARCHING FOR ELEMENTS

In the traversal of the application graph, we have to find for every set of tasks T_i a set of elements E_i that provides enough resources to map all tasks in T_i. In every iteration, we start searching in the topological neighborhood of the elements that were allocated in the previous iteration. From the location of the elements E_{i-1}, a breadth-first search (BFS) is started. When the partial mapping M_{i-1} contains more than one element, we start this search at multiple locations. In the BFS, we try to match the communication infrastructure of the platform to the structure of the task graph, by taking the direction of communication channels between tasks into account. In this search, we keep track of the distance between a newly discovered element and the origins of the BFS, to estimate the cost of the communication routes.

Once sufficient elements are found to map the tasks in T_i, a single additional expansion step (step 2) is performed. This results in a set of candidate elements that is likely to contain more elements than the tasks in $t \in T_i$ require. Without this mechanism, the greedy approach would mainly facilitate an optimization objects towards minimal communication distance.

Up to this point, we described a search method that reduces the larger mapping problem into smaller sub-problems. We still have sets of tasks and elements, but

they are much smaller than the entire application or platform. For each task $t \in T_i$, an element $e \in E_i$ has to be selected. Due to resource constraints, not all solutions are feasible; additionally, we want a solution that performs well for our optimization criteria. The following section describes this assignment problem in more detail.

3.2.2 Assigning Tasks to Elements

The sub-problems we have to solve, are instances of the generalized assignment problem (GAP). A GAP describes a problem where a number of items have to be placed in a number of bins. When both the size and the cost of the items vary over the bins, the problem is 2-approximable [17]. When the GAP has only one bin, the problem reduces to a knapsack problem. In our case, we consider elements to be bins with the resource capacities being the size of the bins. The tasks are the items that have to be placed in those bins, such that the resource requirements are met, and a minimum cost is achieved. In [21], an efficient (approximation) algorithm for the GAP is presented, with a time complexity of $O(|E| \cdot (|T| + k(T)))$, where $k(T)$ indicates the time complexity of a subroutine that solves (single bin) knapsack problems. This algorithm guarantees a $(1 + \alpha)$-approximation solution, where α is the approximation ratio of the knapsack subroutine. These characteristics state that both the quality and time complexity of this approach mostly depend on the knapsack solver. Furthermore, note that if the knapsack subroutine solves the knapsack problem to optimality, also the overall approach does this for the GAP.

Adopting the approach of [21], we process the set of the elements E_i that were discovered in step 2. Iteratively we consider the elements $e \in E_i$ and we calculate for each $t \in T_i$ the cost $cost(t, e)$ of mapping task t to element e. We put these values in a vector c^2 of length $|T_i|$. Another vector c^1 contains the cost of the best known mappings of task t to the already considered elements, initially set to very large values. We pass both vectors to a knapsack routine that selects for the current element $e \in E_i$ a subset of tasks with a maximum cost reduction compared to the current assignment specified by c^1. When element e picks a task t, the cost of that combination is updated in c^1_t. Obviously, we only consider remapping a task t, if the cost reduction $c^1_t - c^2_t$ is positive. Because unmapped tasks have infinite cost, picking a yet unmapped task is always more beneficial than remapping a task to another element. This procedure is now illustrated with an example.

Example 3.2.1 ⌊Iteratively solving GAPs⌉ *Figure 3.6 shows a set of tasks that have to be mapped to three elements, e_0, e_1 and e_2. Starting with e_0, the knapsack routine selects the most beneficial subset of tasks in terms of cost. To simplify the example, the capacities of the elements and the size of the tasks have been left out; instead, we assume that a maximum of two tasks can be assigned to any single element. Element e_0 selects tasks t_1 and t_m; those tasks are marked grey as such. Next, the knapsack routing is applied to element e_1. Task t_1 is incompatible, represented with infinite cost in c^2. The best choice for e_1 is to select tasks t_0 and t_m. As the last element of the first iteration, element e_2 may select t_0 with an increased cost of 6, it may select t_1 with a cost reduction of 1, or t_m with an increase in cost of 3. Hence, the best choice for e_2 is*

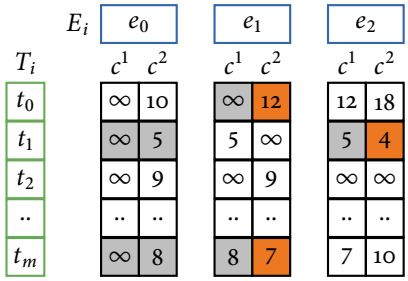


Figure 3.6: Solving the assignment problem of the tasks T_i to the elements E_i. The rightmost gray block of each task t is part of mapping M_i.

to select only t_1. As a result the task is removed from element e_0 and assigned to e_2. Note that due to the move of task t_1, resources have been freed up at element e_0. In a next iteration, element e_0 may therefore select task t_2 as the next best choice. The final assignment of the first iteration is marked with orange.

If we sort the tasks in T_i on increasing cost while calculating c^2, the knapsack routine can be implemented with a linear traversal of the list of tasks. This gives a time complexity for a single knapsack procedure of $k(T) = \Theta(|T| \cdot \log(|T|))$. We perform this procedure for every $e \in E_i$. The time complexity of the combined algorithm to solve the GAP is, therefore, $\Theta(|E| \cdot |T| \cdot \log(|T|))$.

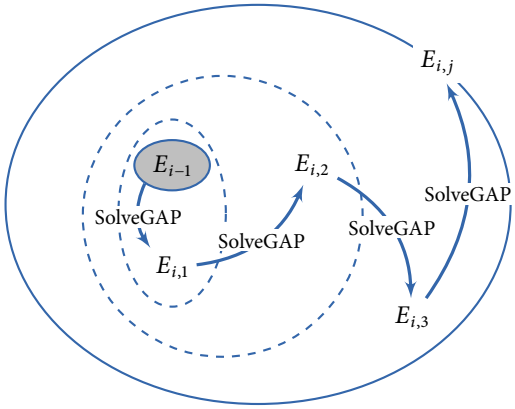


Figure 3.7: Starting from the elements of the previous iteration E_{i-1}, the set of candidate elements E_i is expanded, until a feasible mapping M_i is found.

The procedure explained in the example solves the GAP by considering every combination $\langle e, t \rangle \in E \times T$ only once, greedily selecting the best options. A favorable task may be considered by multiple elements, moving the task around to the best 'bin'. Unfavorably tasks may not even be selected (at first). As a result, the GAP

solver may return only an incomplete mapping M_i. In this case, multiple iterations are required in case some tasks in M_i have not been assigned to elements. In subsequent iterations, elements may reconsider previously unselected candidate tasks. Moreover, if insufficient elements E_i were supplied to map every task, meaning that at some iterations no improvement of M_i occurs but still not all tasks from T_i are assigned, then the GAP solver is invoked again, but with a larger set of elements E_i. Figure 3.7 shows such a growth of the set of elements E_i, until all tasks in T_i have been assigned to an element $e \in E_i$. We retain the mapping and associated cost of the previous iteration while expanding the set of elements E_i. The newly added elements may select yet unmapped tasks, or may steal already assigned tasks, freeing up resources at the lightened elements. Note that when the cost function depends on mapping M_i, it must be re-evaluated every time M_i changes, resulting in an increased complexity. The load of a (processing) element is a factor that exemplifies such a dependency. Otherwise, the cost function can be implementation with a constant-time lookup table.

Summarizing, with step 1 and step 2 the mapping problem is decomposed into sub-problems (divide), while step 3 solves each sub-problem (conquer). The number of sub-problems depends both on the application structure, as well as the number of (I/O) constrained tasks (having one mapping possibility). This procedure is not explained with an example.

Example 3.2.2 ⌊BFS2GAP mapping procedure⌉ *Figure 3.8 presents an illustrative example used with the proposed heuristic; in the first step $i = 0$, we take task t_d in the application graph as a starting point. Assume that task t_d is constrained to the location represented with a blue circle in the 5x5 meshed platform (at row 3, column 4). The filled black circles indicate that those elements in the platform are fully occupied; for simplicity reasons, we allow a single task per element in the platform. At the end of iteration $i = 0$, we have $T_0 = \{t_d\}$ and $E_0 = \{e_{(2,3)}\}$ (using a coordinate system to identify the elements).*

In the next iteration $i = 1$, the application graph yields a set of tasks $T_1 = \{b\}$. Searching for candidate elements, starting with $E_{1,0} = E_0 = \{e_{(2,3)}\}$ gives $E_{1,1} = \{e_{(2,3)}, e_{(3,3)}\}$ and $E_{1,2} = \{e_{(2,3)}, e_{(3,3)}, e_{(3,2)}, e_{(3,4)}, e_{(4,3)}\}$. For the purpose of the example, let task b to assigned to element $e_{(3,3)}$.

Continuing with iteration $i = 2$, the next subset of tasks $T_2 = \{t_a, t_e\}$ is used. A set of elements E_2 is gathered as candidates for the mapping of the tasks in T_2. As a result, task t_a is mapped to element $e_{(3,4)}$ and task t_e is mapped to element $e_{(4,3)}$. The mapping is complete after 4 iterations, which is shows in Figure 3.8e.

Unconstrained applications For the special case, when T_0 is initially empty, a starting point in the application has to be defined. In mapping approaches it is common to select a difficult task, e.g. one with high communication requirements. However, we choose to pick a 'simple' task and try to assign it to a 'difficult' element

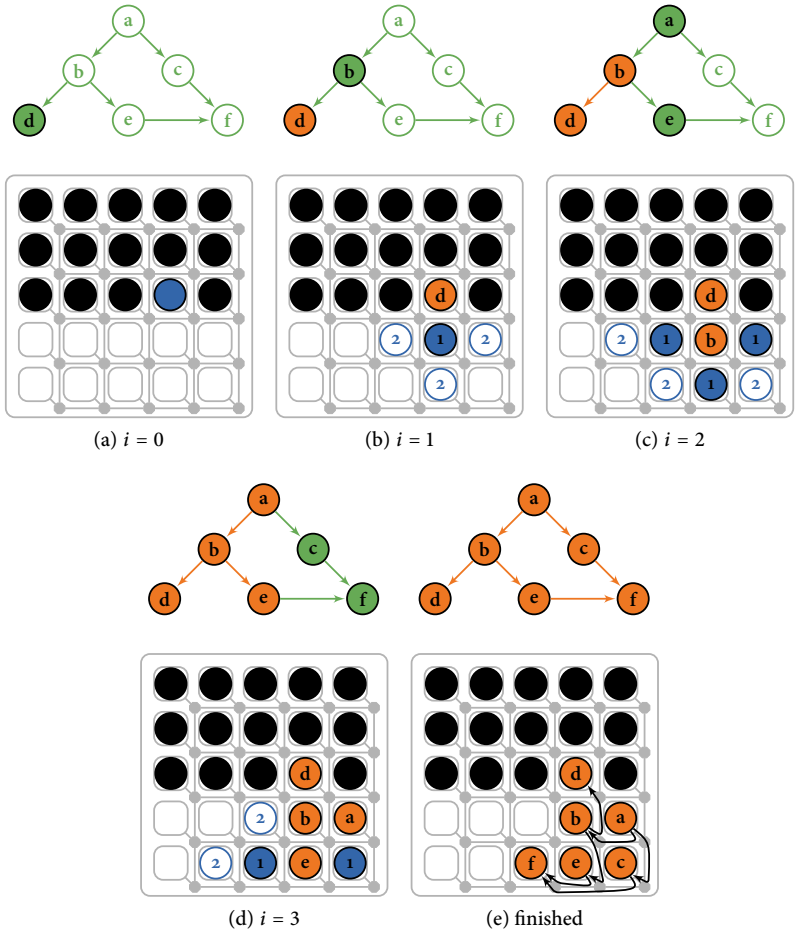


Figure 3.8: Search state after each iteration in BFS2GAP, where gray nodes denote mapping M_{i-1} and dashed nodes compose mapping M_i.

in the system. Elements are considered to be difficult when they tend to become isolated; i.e. when the external resource fragmentation increases. To reason about isolation, we define the *degree* $d(v)$ of a vertex v to be the number of edges connected to vertex v. To reduce external fragmentation of processing elements, we select a task t_0 with a degree $d(t_0)$ that is the lowest in the task graph, indicated with $\tilde{d}(T)$. For this task, we search an element $e_0 \in E$ that is likely to become isolated later on, when it is not used now. Using $\langle t_0, e_0 \rangle$ as M_0, we continue with the three algorithmic steps described in this section. This way, we prevent situations where computational resources are unreachable due to the lack of communication resources.

3.2.3 THE ALGORITHM

Listing 1 provides the implementation of the mapping heuristic, named `BFS2GAP`. The initial mapping M_0 is initialized at line 2 using the tasks $t \in T$ that are constrained to a single element $e \in E$. In the special case of unconstrained applications (when M_0 is empty), M_0 is determined at line 4. Then for each $T_i \subseteq T$, the procedure `SolveGAP` is invoked with T_i and $E_{i,j}$ (line 6-14). On line 13, the set of candidate elements $E_{i,j}$ is extended with each iteration. Note that the `SolveGAP` subroutine is executed at least once more than strictly required to compensate for greedy optimization. This allows for the `SolveGAP` subroutine to do another pass over the $E_{i,j} \times T_i$ combinations. Referring back to Figure 3.6, element e_0 initially selected task t_1. Due to the fact that task t_1 later moved over to element e_2, again resources at e_0 may have become available. This may enable e_0 to select another task, for example task t_2. Optionally, the `SolveGAP` subroutine can be invoked repetitively until the assignment no longer changes. This may improve the quality of the solutions at the cost of increased computational effort.

The inner loop (line 12-21) is terminated when all the tasks $t \in T_i$ have been mapped in the previous iteration $(j-1)$, or when the set of candidate elements $E_{i,j}$ can no longer be extended. The procedure fails when the set of candidate elements is insufficient to map the tasks T_i (line 23). The time complexity of the `BFS2GAP` procedure is $\Theta(ij \cdot |E| \cdot |T| \cdot \log(|T|))$.

Backtracking

Multiple decisions have to be taken in order to solve a run-time mapping problem instance. Only specific combinations of decisions lead to a feasible solution. Each such decision is, however, only a specific point in the search space. If the decision process of a problem can be executed in an iterative manner, a search space can be represented with a graph, in which the nodes define a partial set of decisions that have been taken out of the total number of decisions that compose the problem at hand. An edge between two nodes represents a decision, which can be associated with cost. A problem can be modeled as a tree, where the root node is the starting point in the decision process where no decisions have been taken yet. The goal is to arrive at some leaf node by taking as many decision as required to solve the problem, such that the leaf node represents a feasible solution, preferably with minimal cost. Backtracking is the mechanism that allows the decision process to revisit decisions made higher up in the tree. The heuristic described in this chapter is of an iterative manner, but does not employ any form of backtracking. Techniques exist that work on top of such heuristics to provide a systematic way of traversing a search space [132].

A complete search approach ensures that every node in the search space is considered. Typical problem instances of the run-time mapping problem exhibit a large number of symmetries in their search space. Symmetrical structures in the problem are difficult to prune, causing the algorithm to evaluate each of them. This may lead

Algorithm 1 BFS2GAP

1: **procedure** BFS2GAP($A = \langle T, C \rangle, P = \langle E, L \rangle$)
2: $M_0 \leftarrow \{\langle t, e \rangle \mid t \in T, e \in E, \text{MappingConstraints}(t) = e\}$
3: **if** $M_0 = \varnothing$ **then**
4: $M_0 \leftarrow \{\langle t, e \rangle \mid t \in T, e \in E, d(t) = \tilde{d}(T), \min_E \text{Costs}(t, e)\}$
5: **end if**
6: **for** $i \leftarrow 1..\infty$ **do**
7: $T_i \leftarrow \{n \mid n \in N_i(T_0)\}$
8: **end for**
9: **for all** $T_i \subseteq T$ **do**
10: $E^+_{i-1} \leftarrow \{e_1 \mid \exists \langle t_1, t_2 \rangle \in C, \langle t_1, e_1 \rangle \in M_{i-1}, t_2 \in T_i\}$
11: $E^-_{i-1} \leftarrow \{e_1 \mid \exists \langle t_2, t_1 \rangle \in C, \langle t_1, e_1 \rangle \in M_{i-1}, t_2 \in T_i\}$
12: **for** $j \leftarrow 1..\infty$ **do**
13: $E_{i,j} \leftarrow \{n \mid n \in N_j(E^+_{i-1} \cup E^-_{i-1})\}$
14: $M_{i,j} \leftarrow \text{SolveGAP}(A, M_{i,j-1}, T_i, \bigcup_{j'=1}^{j} E_{i,j'})$
15: **if** $E_{i,j} = \varnothing$ **then** ▷ set E is not extended
16: **break**
17: **end if**
18: **if** $T_i \subseteq T(M_{i,j-1})$ **then** ▷ all tasks are mapped
19: **break**
20: **end if**
21: **end for**
22: **if** $T(M_{i,j}) \subset T_i$ **then**
23: **fail**
24: **end if**
25: **end for**
26: **end procedure**

to long execution times. Due to non-functional requirements regarding the allowed computation time, a complete search approach is considered to be inapplicable. In [132], an incomplete search technique called *beam stack search* is presented, which is considered as a backtracking mechanism on top of the heuristic described in this chapter. Preliminary experiments showed that the computation time using this search strategy exceeds the reasonably allowed response time. This can be attributed to both the symmetry in the problem as well as the lack of feedback in case of infeasible search areas.

3.3 Empirical validation

To test the proposed heuristic thoroughly, we aimed for a large set of applications. As the availability of models of real world applications is low, we resort to using synthetic datasets. An in-house developed application generator, similar to TGFF [28], is used to generate annotated application graphs. In this tool, the structure of an application can be specified with a number of input, internal, and output tasks. Also the maximum in-degree and out-degree of tasks gives direction to the generated

Algorithm 2 SolveGAP

```
1: function SolveGAP(A, M_{i,j}, T_i, E_i)
2:     for all t ∈ T_i do
3:         c_t^1 ← ∞
4:     end for
5:     for all e ∈ E_i do
6:         for all t ∈ T_i do
7:             c_t^2 ← costs(t, e)
8:         end for
9:         T_i' ← sort(T_i, c^2)                ▷ sort T_i on non-decreasing c^2 w.r.t. e
10:        for all t ∈ T_i' do
11:                                             ▷ verify if e still has the resources to sustain t
12:            fits ← sustainable(M_{i,j}, t, e)
13:            if fits ∧ c_t^1 − c_t^2 > 0 then
14:                c_t^1 ← c_t^2
15:                M_{i,j} ← M_{i,j} ∪ ⟨t, e⟩    ▷ any previous assignment of t are invalidated
16:            end if
17:        end for
18:    end for
19: end function
```

communication structure. Independent from the application structure, tasks are annotated with bounded random resource requirements. We generate applications that are either *computational intensive* or *communication oriented*. Tasks of applications in the first set use between 70% and 100% of the element's resources, and tasks in communication oriented applications use between 10% and 70%. This allows for communication oriented applications to time-share elements, eventually resulting in communication bottlenecks. Within these categories, we sort applications based on their size, namely *small* (≤ 5 tasks), *medium* (6-10 tasks) and *large* (11-16 tasks) applications. Table 3.1 shows the six datasets, each initially containing 100 applications. To filter out any extraneous samples, we remove applications from the dataset that cannot be mapped, even if the platform is empty. The third column (size) in Table 3.1 lists per dataset the number of applications remaining after the filtering.

The BFS2GAP algorithm presented is implemented in the C language [TDtB:1], and is tested on a 200 MHz ARM926Ej-S processor, running a Linux 2.6.28 kernel. The CRISP hardware is used as the platform providing the resources. Chapter 4 describes this platform in more detail, when we consider real-world applications and use-cases.

Per dataset listed in Table 3.1, we generate 30 random sequences of the applications, containing no duplicate entries. We evaluate the mapping heuristic, by iteratively adding an application to the platform. The available resource capacities reduce with every successfully mapped application. Already relatively early in the sequence, most platform resources are already allocated, probably resulting in rejection of the

Table 3.1: Dataset Characteristics and Failure Percentage per Phase.

Dataset Characteristics		Size	Failure Distribution Mapping	Routing
Communication	Small	97	1.05%	98.95%
Communication	Medium	57	15.33%	84.68%
Communication	Large	22	3.45%	96.55%
Computation	Small	99	95.36%	4.66%
Computation	Medium	94	87.28%	12.72%
Computation	Large	96	61.95%	38.05%

remaining applications. To verify that this is indeed the case, all applications in the sequence are mapped, regardsless of the mapping result of previous applications. With this procedure, we mimic scenarios in which multiple application share the platform resources, running side-by-side, and scenarios where the availability of resources may differ from design-time expectations.

3.3.1 Evaluating performance of the heuristic

For successful resource allocation attempts, the average execution time is plotted in Figure 3.9. The measurements are categorized on the number of tasks in the application. The required time to map an application is split into the time attributed to task mapping and time needed for communication routing.

The experiments are designed such that failure is expected later in the application sequence. The reason for the failure to map an application is listed in Table 3.1; namely, task mapping or communication routing. The results show that a lack of communication resources generally causes the rejection of a communication oriented application. Computation intensive applications are mostly rejected in the mapping part due to the lack of computational resources such as central processing unit (CPU) time or memory. In the dataset with large, computation intensive applications, the communication resource requirements also become significant, resulting in more failures in the routing part.

3.3.2 Evaluating optimization objectives

To qualify the mapping cost function, we investigate the influence of the mapping objectives. We optimize towards communication minimization, fragmentation reduction, and a combination of both objectives. Also, we disable the cost function, indicated with 'None'. The resulting mapping then depends on the communication minimization that is inherent to the resulting first-fit search method.

Figure 3.10 shows the allocated number of links per communication channel, represented by the points in the figure and the left axis. The mapping success rate is plotted with solid lines and relates to the right axis. After the 15^{th} application, the mapping success rate drops below 20%. The applications that are still admitted,

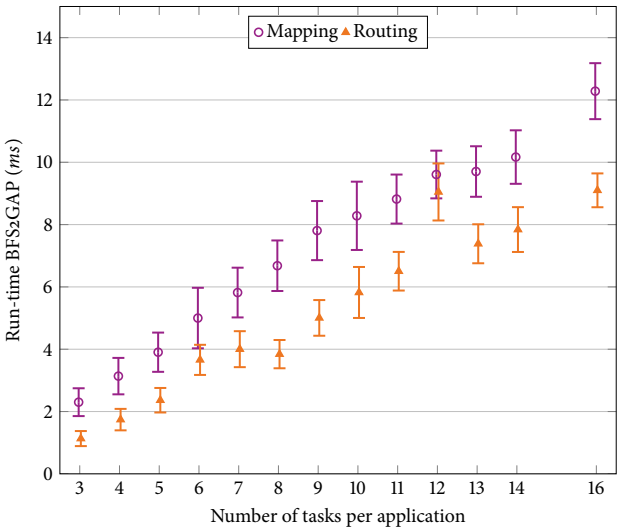


Figure 3.9: Execution time of BFS2GAP for the applications in the synthetic datasets.

are allocated less communication resources compared to applications earlier in the sequence. This indicates that an application is only admitted to an almost saturated platform, if an area with adjacent elements is still available. This would make a case for an optimization objective that considers fragmentation. However, using an optimization objective that does not consider resource fragmentation gives a higher mapping success rate. A possible explanation is that the focus on fragmentation reduction results in premature optimization which is not paid back in the long term.

Figure 3.11 shows the external resource fragmentation of the elements in the platform, in relation to the progression of the application sequence. Furthermore, the success rate is repeated in this graph. The cost function behaves as expected, as the fragmentation reduction objective (plotted in yellow) results in the lowest fragmentation compared to the other optimization objectives. Overall, the fragmentation converges to 30% and the mapping success rate converges to 10%. Aiming at fragmentation reduction (Figure 3.11) increases the average communication distance (Figure 3.10), resulting in a lower mapping success rate.

3.3.3 Discussion

The heuristic presented in this chapter gives results within tens of milliseconds for applications up to 16 tasks, measured on a low-end embedded processor. While the response time is in line with expectations, additional computation time is expected if the heuristic is extended with a backtracking mechanism. Without it, the main weakness in the algorithm is the divide and conquer approach itself. The tasks

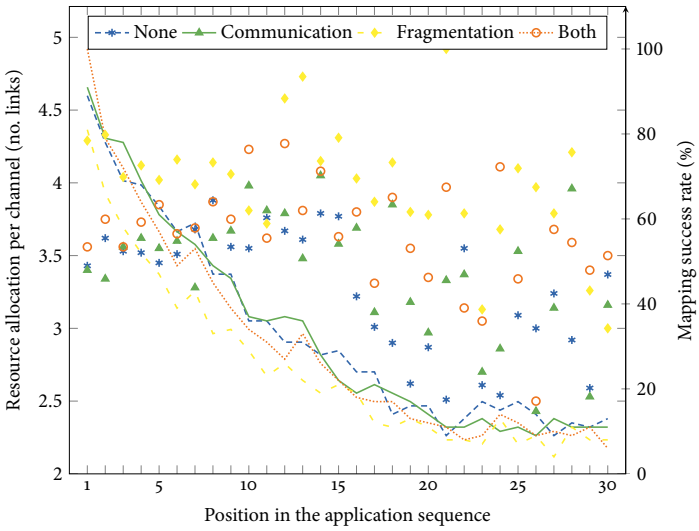


Figure 3.10: Average number of communication links allocated per channel.

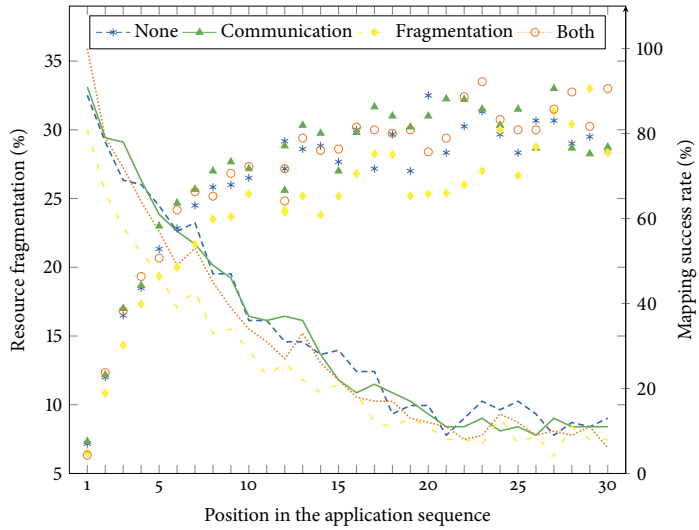
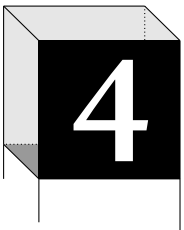


Figure 3.11: External fragmentation of platform resources, averaged over all datasets, using various optimization criteria.

in the application are subdivided into sets based on the communication topology. When the first mapped tasks are assigned to resources in vastly different areas in the platform, possibly restricted by mapping constraints, it might become difficult later on to 'connect' the different parts of the application. The lack of a global view on the problem may cause strongly connected components in the application graph to become isolated.

The empirical validation of this chapter is done with a synthetic dataset of applications. The results observed may be subject to a certain degree of mismatch between applications and the target platform. The next chapter describes the CRISP platform used for validation in greater detail, together with some real applications specifically developed for the CRISP platform.



Case Study 1: The CRISP project

Abstract – *Within the Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project, a scalable and dependable reconfigurable many-core system concept has been developed. Scalability of the architecture is enabled by the interconnect, which transparently extends the network-on-chip, off-chip and even off-board. The interconnect requires explicit configuration, using routing tables at various locations in the platform. This feature allows the system to control the resource allocation of applications, and to configure the hardware accordingly. This configuration can be done both manually, and with a run-time resource manager, which provides distribution transparency to the user and increased robustness against hardware faults.*

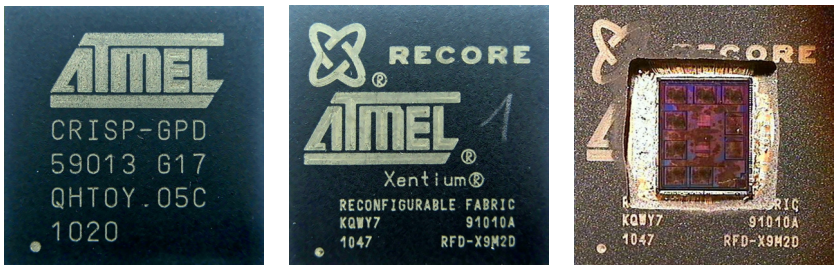
4.1 The General Stream Processor

Within the Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project, a scalable and dependable reconfigurable many-core system concept has been developed, named the general stream processor (GSP). To investigate the envisioned scale of future systems, within the CRISP project the GSP is prototyped with an aggregation of multiple integrated circuits (ICs). This instance of the GSP consists of one general purpose device (GPD) and 5 reconfigurable fabric devices (RFDs) together with an FPGA subsystem for data generation and output capturing. Figures 4.1a and 4.1b show the two types of ASICs manufactured within the project. Each of these chips and the components inside will be further described in this section.

4.1.1 Reconfigurable fabric device

The RFD is the main constituent of the CRISP platform, manufactured in UMC 90nm CMOS technology, occupying around 44mm², and runs at 200 MHz. The RFD contains 9 Xentium® fixed-point DSP cores [91] (XE_0 to XE_8) in Figure 4.2) and two on-chip memories (MEM_0 and MEM_1) in a tiled architecture. Each Xentium® core contains 16 KB data memory and 8 KB instruction cache, and each memory tile

Parts of this chapter have been published in [TDtB:3, TDtB: 5, TDtB: 6].



(a) GPD (b) RFD (c) Die-photo RFD

Figure 4.1: Chips designed and manufactured within the CRISP project.

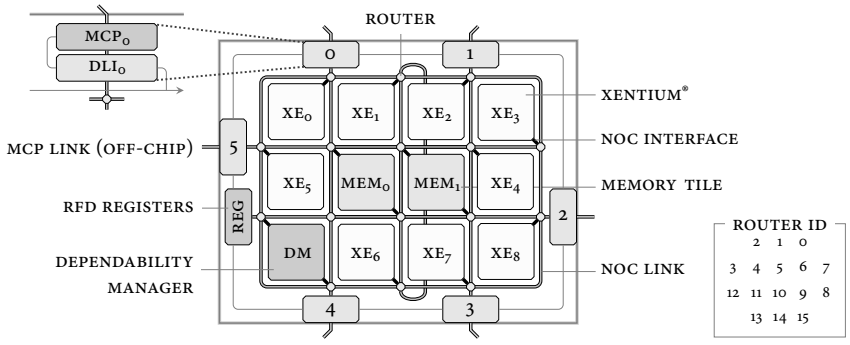


Figure 4.2: The reconfigurable fabric device developed within the CRISP project.

combines 64 KB of static random-access memory (SRAM) (divided in two banks) with flexible address generation units. The address generation units can be used to reorder a datastream or use the tile as a first-in first-out (FIFO) buffer. The tiles within the RFD are interconnected by a 16-router GuarVC NoC [118] mesh, which in turn is connected to adjacent chips with 6 multi-channel port (MCP) ports (connectors 0 to 5 in Figure 4.2). The routers are represented by the small hexagons in Figure 4.2 and the identifiers of the routers are given in the bottom right of Figure 4.2. The MCPs are combined with die link interfaces (DLIs) to extend the NoC transparently off-chip. The dependability manager (DM) is a special tile, connected to two NoC routers, to perform online dependability testing of the Xentium® cores. Finally, a register bank (REG) is available to configure the DLIs and to configure the clock frequency of the RFD.

Network-on-Chip

A NoC is made up of a set of routers interconnected with a limited number of *physical channels* between each router. A physical channel can be shared between multiple connections using timeslots. Various scheduling policies can be used to

assign timeslots to connections. In the GSP, time-sharing of the physical channels is implemented with the concept of *virtual channels*. The router arbitrates over a physical channel, splitting the available bandwidth between the allocated virtual channels. This arbitration can be implemented for example with round-robin, optionally with different budgets per virtual channel. Due to hardware restrictions, only a limited number of virtual channels per physical channel is provided. The 'GuarVC' NoC routers used in the RFD support up to 4 virtual channels per physical link, which is a reasonable trade-off between router area and communication throughput [70, 71].

The GSP implements the GuarVC NoC protocol to transport data through its heterogeneous interconnect. This protocol uses *source routing*, which allows a sender to specify the route a packet takes through the network. Such a packet is broken into smaller pieces known as flow control digits (FLITs), consisting of a header FLIT, one or more data FLITs and a tail FLIT. Whenever a router receives a header FLIT on the incoming physical link, the packet is stored into the virtual channel buffer corresponding with the information contained in the FLIT. A FLIT can only be transferred if the virtual channel buffer at the destination router is not full; otherwise, another virtual channel with pending traffic is served. A weighted round robin arbiter allows at each turn a flit of one of the non-empty buffers to proceed through the crossbar. The header FLIT contains for each router to be traversed the switch path that needs to be taken; FLITs arriving at any link in Figure 4.3 can be routed to all other links but the incoming link (a U-turn is not possible). The router removes this information from the header; the size of the header then reduces with each hop in the network. A correctly routed packet no longer contains any routing information in its header once it reaches its destination.

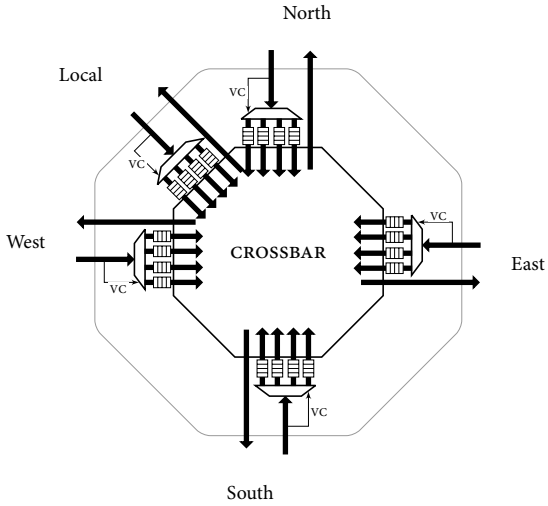


Figure 4.3: GuarVC router

The length of a communication route encoded in the NoC protocol header is limited. Longer communication routes can be constructed by chaining multiple short routes. Whenever a route encoded in a packet header ends at a DLI, the route is extended by the route programmed in the DLI. The routing tables of the DLIs consist of two levels; one level is to address other RFDs reachable through the MCPs, while the other level is used to address resources on the local chip. This configuration ensures that the NoC is transparently extended off-chip in order to combine all RFDs in a single interconnect. The routes configured in the DLIs are calculated by the run-time mapper.

The NoC of the GSP supports both a best-effort packet-switched mode, and provides guaranteed bandwidth through circuit-switched connections. The resource manager developed in the CRISP project routes communication channels between tasks using circuit-switched connections, allocating a single virtual channel per physical link on the communication route. This choice results in a restriction that no more than 4 communication channels can be routed over a single network link.

The tiles of the RFD are connected to the routers with a network interface that provides the abstraction of a master/slave memory bus. Tiles such as the DSP and DM have a master (network) interface each containing a routing table. Accesses to off-tile memory regions are then encapsulated in a common request/response protocol. The header of the protocol takes a pair of routes from the routing table to define the forward and return path through the interconnect. When a memory range corresponding to a routing table entry is accessed, a connection is established over the forward path. The addressed peer network interface will subsequently respond with the requested data using the return path. The routing tables provide a means to (re)configure the way code executed inside the tile interacts with resources outside the tile. The configuration of the routing table entries should thus be consistent with the application, which assumes that it is able to access specific resources through those routes. The resource manager can exploit the routing tables to (re)configure the view of each tile to the 'outside' world, providing *location transparency* to the largest part of the GSP. A task can thus be assigned to any compatible tile, as long as the routing tables of other tasks that communicate with it are configured accordingly.

Dependability Manager

Safety critical systems often have built-in fault detection mechanisms. Replacing faulty components manually is not always an option due to inaccessibility of the hardware, as in space or hazardous environments, or due to the tough requirements on the availability of the system. If, upon detection of a fault, the faulty components can be localized and isolated, the system might continue its operation while circumventing the hardware fault. To be able to integrate this level of dependability management, some testing infrastructure is built into the platform. Figure 4.2 shows the DM, which is used to perform built-in self tests (BISTs) and scan chain tests simultaneously using up to three Xentium® cores [72].

The degree to which such a system may repair itself highly depends on the flexibility exposed by the applications running on the system. A static resource assignment, often performed at design-time, limits this flexibility. Our resource manager is able to incorporate and benefit from the dependability features of the platform. In Section 4.2.1, we demonstrate that system tests and monitoring applications can be run alongside 'normal' applications.

4.1.2 General purpose device

Next to real time computational work, embedded signal processing demands configuration and control. Therefore, the platform incorporates a general purpose device (GPD), which is a customized micro-controller [6] with an ARM926EJ-S processing core, running at 200 MHz. The GPD is loaded with a Linux-based operating system for which a driver is developed to access the NoC. Figure 4.4b shows the location of the GPD relative to the RFDs.

4.1.3 Hardware verification board

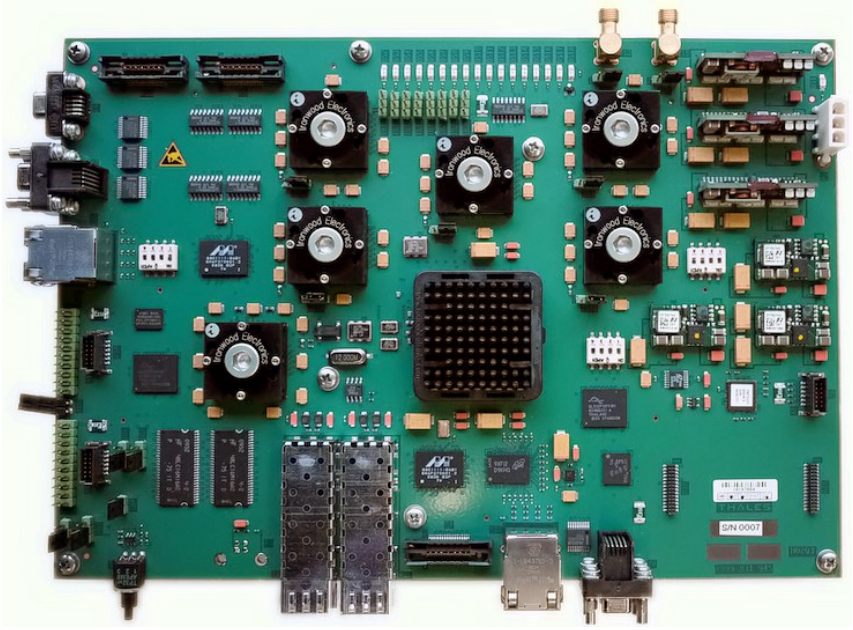
Like the NoC is used to connect all components on-chip, an off-chip network is used to combine RFDs and GPDs into a scalable architecture. MCP ports on the RFD function as transparent bridges between the NoCs of different RFDs. A hierarchical addressing scheme is used to reduce the routing overhead. With the same technology, multiple boards can be connected together as well[1], resulting in a heterogeneous hierarchical network of resources.

Figure 4.4a shows the printed circuit board (PCB) that is used for the verification of the GSP concept. Besides the GPD, the board can support up to 5 RFDs. Figure 4.4b identifies the various components mounted on the board.

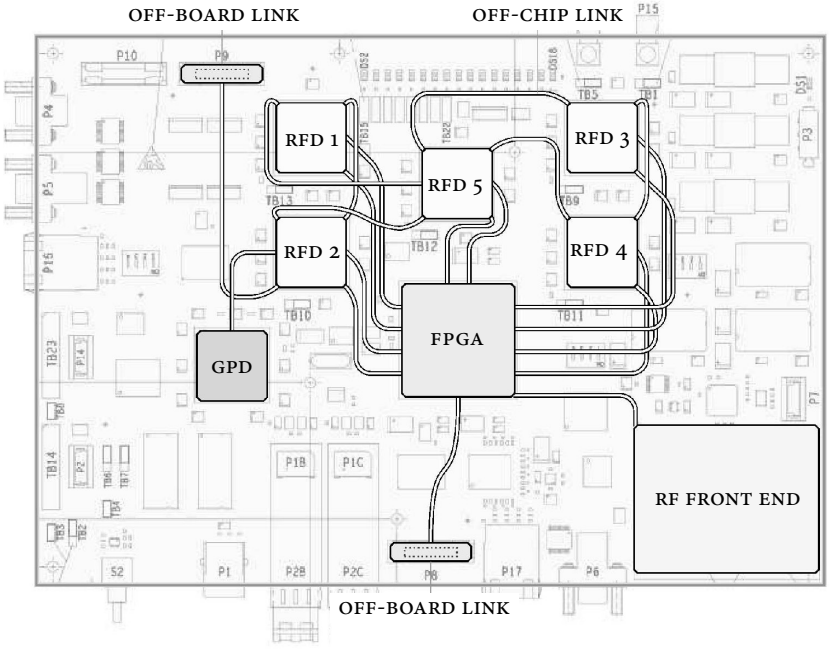
4.2 Software stack

In this section, all entities are described that are required to instantiate a run-time mapping system as depicted in Figure 1.3. The software stack deployed onto the GPD contains a model of the platform, a run-time mapper and a board support package. Applications loaded onto this software stack come with a model of the application specified in the format as required by the run-time mapper.

[1] In an experimental setup, three boards were connected into a single platform. No changes were required to the software stack to operate this platform. The performance-wise scalability of centralized management proofed to be limited.



(a) Hardware verification board.



(b) Schematic view showing the various components.

Figure 4.4: A GSP instantiation of one GPD and 5 RFDs.

The GPD runs a Linux kernel and software stack to provide end-users with the interaction with and control of the GSP. A device driver has been implemented to communicate with the NoC [112]. The user receives a handle to a specific resource on the platform by sending routing information to the device driver. Common file-operations can then be used to communicate through this handle with the requested resource. This allows for a rich-featured platform support library that is independent of the specific location of resources. Figure 4.5 shows an example, in which routing information is passed to the function `c2c_open`, which returns a file descriptor (handle). A route is encoded by two bits per hop through a router on the communication path. Whenever the current hop is encoded by the value 0, the incoming packet is routed to the clockwise next logical output port, relative the input port of the router. A non-zero value indicates the number of ports that need to be skipped, allowing a packet to be routed to any of the output port other than the input port. A packet leaves a router with one hop less encoded in the protocol header, reducing the length of the header as the destination nears. For example, the route specified in Figure 4.5 takes MCP 5 in Figure 4.2 as a starting point, and via a 'west-east' and 'west-local' hop ends up in memory tile 0. The function `c2c_write_value` then writes the value `0x1` to the address at offset `0x200` in the memory tile, without knowing the type or location of the resource that is being accessed.

```
// Declarations
struct gv_header fw, rt;
int h;
// Set the 'forward' NoC route to [(WEST->EAST),(WEST->LOCAL)].
gvheader_init_with_route(&fw, "WEWL");
// Set the 'return' NoC route to [(LOCAL->WEST),(EAST->WEST)].
gvheader_init_with_route(&rt, "LWEW");
// Open a connection to NoC resource using the specified route.
h = c2c_open(&fw, &rt);
// Using the handle, write value '0x1' at offset '0x200'.
c2c_write_value(h, 0x1, 0x200);
// Close the resource corresponding with handle 'h'.
c2c_close(h);
```

Figure 4.5: Example usage of the C2C device driver to access the NoC.

Platform discovery and fault detection

For initialization of the system, a platform specification is used that describes the platform elements assumed to be present. The status of each resource in that initial platform specification is set to 'absent', and the communication links to 'faulty'. Iteratively with every RFD, starting with the RFD closest to the GPD (RFD2), the following steps are performed. First, a probing mechanism detects whether the chip is actually present and responding. In the RFD registers (see Figure 4.2), the clock frequency of chip is set (to 200 MHz). Second, similar techniques as described

in [52] and [25] are used to perform functional tests on the interconnect. Links that are found to be fault-free are marked as such in the platform model. After the validation of the interconnect, the routing tables of the DLIs are configured to enable access to other parts of the board.

As a third, after the validation and configuration of the interconnect, the Xentium® cores and memories are examined. Figure 4.2 shows the DM, positioned in the bottom left corner of each RFD. This tile provides hardware support to perform structural tests of the Xentium® cores, using the NoC as test access mechanism [130]. Using the principle of majority voting, triples of Xentium® cores are tested for faults. The platform model is then updated according to the latest test results. The dependability test approach reported in [129] is implemented and supported by the author of this thesis as a common application for the run-time mapping system. These dependability tests can be scheduled periodically to monitor the platform during operation. The set of available resources may change over time, either by changing the status of the elements and links, or by altering the structure of the platform model (reflecting addition or removal of hardware). These dynamics can only be put to use if applications can deal with this uncertainty in resource availability.

4.2.2 Platform model

Resource management requires knowledge about the platform that has to be controlled and maintained. In conventional systems, information about the underlying hardware is provided by a board support package, used to initialize data structures within the operating system kernel and accompanying drivers. Such information mostly concerns the local processing node and its peripherals. Larger systems have to maintain information about the current state of multiple processing nodes and their connectivity. Within a hierarchical platform model, information about the available computation as well as communication resources of multiple nodes can be combined to get a global view of the platform.

We developed a model that describes the amount, type and connectivity of the resources that are provided by the platform. At the top level, the platform is described by a single *element*, which contains *components* and *links* between those components. Within components, a new level of elements is defined. Recursively, each element contains a subset of (more fine-grained) resources that are spatially related to each other. The elements higher in this hierarchical platform definition sum the resources available in their substructure. These *composite* elements allows for the platform model to be queried at various levels of granularity. For example, an element modeling an RFD specifies the availability of 9 Xentium® DSPs, but does not provide their location or interconnectivity. The structure of the platform model we developed is given below as a grammar using regular expressions[1], where

[1] A regular expression is the definition of a pattern, where parentheses are used to group tokens, in which the '|' indicates a *choice* between members of a group; '*' means *zero or more* and '+' means *one or more*.

non-terminals are formatted in italics:

platform → *element*
element → *element_type components links*
incoming outgoing schedule *status*
element_type → (T_{ARM} | $T_{XENTIUM}$ | T_{MEMORY} | ... | T_{FPGA})
link → *link_type components links*
incoming outgoing schedule *status*
link_type → (T_{NOC} | T_{MCP} | T_{EXT})
components → (*element_type element*$^{+}$)*
links → (*link_type link*$^{+}$)*
incoming → (*element* | *link*)*
outgoing → (*element* | *link*)*
status → ($S_{CORRECT}$ | $S_{UNKNOWN}$ | S_{ABSENT} | S_{FAULTY} | S_{FAULTY_CORE} | S_{FAULTY_MEMORY}) timestamp

In our platform model, the structures describing links and elements in the platform have the same representation. Each element in the platform model is of a given *element_type*, and every link in the platform is of a specific *link_type*. Different types are used to distinct between them, but they are modeled, queried and scheduled in an identical manner. Both links and elements have a schedule that is used to account the budgets of tasks and channels that have access to the modeled resource. The *status* of an element or link indicates the condition of the element and a timestamp of the last modification. As an example, the CRISP architecture is described next in terms of this platform model.

Example 4.2.1 ⌊The CRISP 46-core platform⌉ *Figure 4.4a shows the hardware verification board that hosts two kinds of chips; RFDs are used for signal processing, and one GPD controls the platform and the FPGA is mainly used as data generator for predefined test scenarios. Figure 4.4b shows a schematic overview of the platform at the board level. On this level, the platform is described as one element (the board), having three sets of resources as components (GPD, 5 RFDs, and an FPGA), which are connected together using off-chip connections as links. Each of these components can be queried when more details are desired at the level of the ARM and DSP cores. The hierarchy in the model can be exploited for practical representation to the user; for example, '*`/gsp0/rfd2/xe1`*' refers to Xentium® number 1 on RFD 2 within GSP 0.*

4.2.3 APPLICATION MODEL

As defined in the problem formulation of Section 2.1.4, each application is composed of a set of tasks that are connected by channels. Each task must have at least one *implementation* that denotes to which type of elements the task can be mapped. A recipe for *initialization* of the implementation may be defined by using a limited number of predefined operations. Executing a piece of code on a Xentium®, for example, requires the core to be reset and the binary code to be loaded into a memory,

after which the start address of the program is to be written in a special register (mailbox) of the core.

The implementation of a task is not limited to using hardware blocks for processing, but it may also require other hardware, like memories and I/O peripherals. The on-board FPGA, the DM and the RFD registers are examples of fixed hardware components that may also be required by tasks within the application. The framework supports application specific needs to use I/O ports or to configure hardware. A task is linked to other tasks through *ports*; an input port (C_{IN}) is matched with an output port (C_{OUT}) within the same application, having the notion of a *channel*. The amount of resources required (budget) is specified per port and implementation. A more detailed description of an application specification is given below:

application	→ name *task*$^+$
task	→ name *port*$^+$ *implementation*$^+$
port	→ name (C_{IN} | C_{OUT}) budget
implementation	→ *element_type* budget *initialization**
initialization	→ (I_{RESET} | I_{SEND_MAIL} | ... | I_{LOAD_BINARY})

Whenever communicating tasks are assigned to different cores, a route through the interconnect is determined. The routing tables of the cores involved are then configured such, that the index of a port in the channel list maps to the corresponding entry in the routing table. A task then logically communicates with other tasks through application defined channels, whereas the system determines the location of the tasks and the means (routes) of communication.

4.2.4 Run-time mapper

The resource manager developed in the CRISP project works on the granularity of tasks and channels within an application, while being unaware of the application's function. Applications may interact with the resource manager through a limited set of function calls. The application programming interface (API) provided is designed as an event-driven software architecture. This allows the resource manager to order the stream of incoming events. Events are handled first based on their priority, and second in order of arrival. The type of an event determines its priority; for example, releasing resources has priority over allocating new resources. Special applications, such as the dependability test software, may have a special priority that distinguishes them from normal user applications. This distinction is not required, but it allows for prioritizing between the intended functionality of the system and provided services such as the dependability enhancement.

The software stack as described in this section is designed for various applications developed in the CRISP project for validation and demonstration purposes. Some of these applications are described in the following section.

4.3 Applications

Two user applications have been developed within the CRISP project; global navigation satellite system (GNSS) reception [59] and digital beamforming. The GNSS application tracks four satellites to determine the global positioning system (GPS) coordinate of the receiver. The processing can be done on a single RFD. The beamforming application receives data generated by the FPGA, emulating 16 antennas, and requires all five RFDs to generate 8 output beams at a rate of 640 Mbps. In a reduced mode, the data of only 8 emulated antennas is processed to form 4 beams. These application modes are referred to as BEAM8 and BEAM4, respectively.

In the next section, the GNSS and beamformer applications are described in more detail, as these applications are used in Section 4.4 for evaluation of the run-time mapping concepts.

4.3.1 A global navigation satellite system receiver

Today's satellite navigation applications range from cheap receivers embedded in mobile phones to expensive and highly accurate scientific ones. In the CRISP project, the GNSS application is specified and designed to support the existing GPS (U.S. based NAVSTAR Global Positioning System) and future Galileo (European system) signals. The three main blocks in any GNSS receiver are identified to be

1. a radio frequency (RF) front end for analogue signal processing,
2. a digital baseband processing part, and
3. navigation calculus to determine position, velocity and time (PVT) from measured pseudo ranges (distance between satellite and receiver).

Figure 4.6 shows a simplified version of the GNSS application; the entire task graph consists of 17 tasks and 68 unidirectional channels (see Figure B.2 for the graph). After some input processing (IP), two acquisition tasks (Acq 0-1) are looking for satellites. A tracker task (Tra 0-3) is started for each completed acquisition process. The GNSS application is able to solve the PVT of the receiver if four or more satellites are tracked successfully. Thus, the receiver application should always (after an initial acquisition stage) have four or more cores running a tracking process to enable navigation. If a satellite is lost, no satellite data is available until the acquisition procedure is again performed and concluded. The implementation steps towards the CRISP GNSS application are explained in more detail in [59].

4.3.2 A 16-channel beamformer

A phased array antenna is able to electronically track signals originating from one or multiple sources. Beamforming is a signal processing technique to transmit or receive data by digitally changing the directionality of such an antenna. After an analog processing stage (not shown in Figure 4.7), a Hilbert transform $\mathcal{H}$ is used to derive the analytical representation of the digital baseband signal. Then, by means

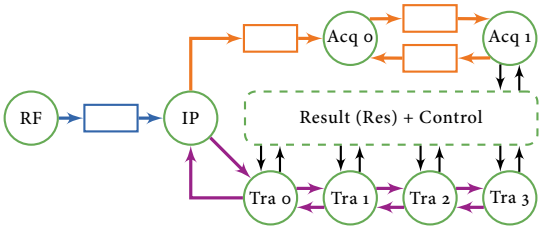


Figure 4.6: Simplified task graph of the GNSS application

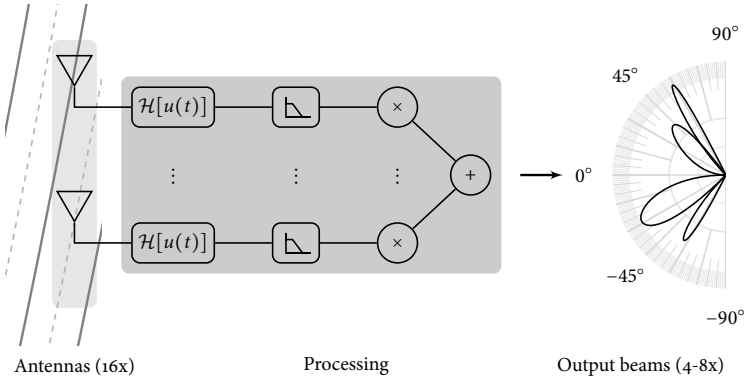


Figure 4.7: Digital beamforming

of a weighted addition of the signals after filtering, the antenna array becomes sensitive to signals from a specific direction resulting in a so-called *beam*. By using multiple weighted additions in parallel, multiple beams are formed. Figure 4.7 illustrates the input, processing and output of such a system. Since in our experiments we only realized the digital processing part (excluding the antenna, analog processing and A/D conversion, data was locally generated on the platform. The FPGA on the GSP shown in Figure 4.4 feeds a recording of antenna data into the platform. The CRISP beamforming application takes as input the signals from 16 or 8 antennas (also referred to as 'channels') to form 8 beams. A reduced mode is available that forms 4 beams out of 8 channels. The processing chains consists of Hilbert and low pass filtering and beamforming. The BEAM8 application mode has 85 tasks and 252 unidirectional communication channels, while the BEAM4 mode (with 8 antennas, 4 beams) contains 53 tasks and 148 channels. The task graphs of these applications are found in Figure B.3 and Figure B.4, respectively. The datastream containing the beam information as output is directed, through the FPGA, off-board for validation and further processing.

4.4 Empirical evaluation

Within the CRISP platform, the applications described in the previous section work on top of the run-time mapping system. The run-time mapping system is evaluated on its functional correctness, its response time and mapping efficiency using three different scenarios. In the first scenario, the CRISP applications are mapped to a *fault-free* chip. Next to this normal operation condition, two scenarios are evaluated in which some parts of the hardware are no longer expected to work correctly. One scenario concerns a single hardware fault, and combined hardware faults are considered in the last scenario. We assume that, in these scenarios, the amount of resources demanded by the application is not changed, such that it requires the affected tasks to be shifted to a different set of resources. In case the platform is either seriously compromised, or its available resources are near depletion, applications may be refrained from execution. The evaluation investigates this breaking point, both from an application perspective and from a hardware perspective.

4.4.1 A fault-free scenario

To evaluate the delay added by the resource manager to the startup of applications, a fault-free scenario is used. The overhead, attributed to each of the stages in the life-cycle of an application are:

1. parsing the task graph, which contains all the information required to start the application, including memory addresses, application code and data and initialization procedures,
2. allocating the resources for the application,
3. starting the application by configuring the requested resources, which consists of initialization of hardware and uploading of binaries (program code and associated data),
4. stopping the application; this deconfigures all resources used by the applications,
5. releasing the resources afterwards.

The first three stages are of most interest when it comes to the delay observed by the user before the application is running. During the experiments, we try to minimize the interference of other (unrelated) activities on the GPD to get accurate measurements. As the algorithm is deterministic, variation in the timing measurements are due to the scheduling and memory allocation performed by the operating system. The reported computation time is an average of 10 runs to compensate for this.

The resource manager runs on the GPD, which contains an ARM926EJ-S core running at 200 MHz. Table 4.1 shows the time required by the resource manager running on the GPD to perform all operations required to start and stop the BEAM8, BEAM4 and GNSS applications. For all three applications, a little over 10% of the time is required to parse the task graph and create the associated data structures.

Then, most of the time is devoted to the resource allocation (the mapping of the tasks and channels to the RFDs). Note that we put as much flexibility in the application as possible; adding constraints to the mapping problem leads to an outcome (admission or rejection) faster due to a reduced search space.

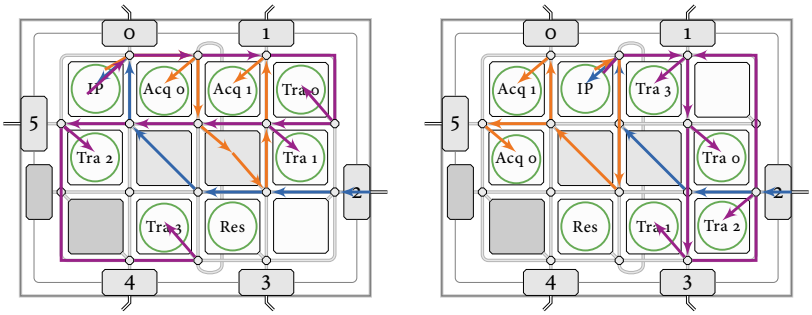
Table 4.1: Time required to start and stop the applications.

	Phase	GNSS	BEAM8	BEAM4
1	Parsing	53.56 ± 3.13 ms	309.8 ± 17.2 ms	149.4 ± 4.7 ms
2	Allocation	318.76 ± 10.31 ms	2818.9 ± 8.6 ms	1768.9 ± 31.4 ms
3	Configuration	43.71 ± 0.44 ms	204.1 ± 8.3 ms	124.8 ± 0.8 ms
4	Deconfiguration	4.5 ± 0.1 ms	14.7 ± 0.1 ms	12.0 ± 0.4 ms
5	Release	11.15 ± 0.25 ms	76.0 ± 0.3 ms	76.0 ± 0.3 ms

*The plus-minus sign (±) denotes one standard deviation.

The GNSS receiver

The GNSS application of section 4.3.1 is mapped by the software designer to the resources of a single RFD. Figure 4.8a shows (a part of) the manual mapping; for clarity, not all communication channels are drawn. The colors of the connections that are marked match with the edges of the task graph in Figure 4.6. An input datastream is taken from the RF front end (located bottom right of Figure 4.4b), which is routed through the FPGA to I/O port number 2. Within the memory tile, a FIFO buffers the datastream, which is in turn read by the input processing task (IP). After decimation, the IP-task first forwards the datastream through another FIFO buffer to the acquisition chain (Acq 0,1), and then writes the same samples in the local memory of the first tracker task (Tra 0). The result processing and application control part are executed on the ARM processor in the GPD, which is outside the scope of Figure 4.8.



(a) Manual mapping on a fault-free chip.

(b) Computed mapping on a fault-free chip.

Figure 4.8: GNSS application mapped to a single RFD.

Figure 4.8b shows the mapping of the GNSS application to a fault-free RFD as proposed by the resource manager. The generated mapping is more efficient, in terms of communication resources, than the manual mapping; the number of links used per communication channel is on average 6.4±1.4 versus 6.8±1.8 links in the manual mapping.

Mapping a beamforming application

The beamforming application is a regularly structured application, specifically designed for the GSP. The application consists of four filter components that process the antenna data in a data parallel fashion, and one combining component. Each component is designed to execute on its own RFD. The four 'filter' RFDs are symmetrical, as shown in Figure B.3 and Figure B.4. Most of the actual beamforming is then performed on the fifth RFD. The need for automatic resource management is emphasized by the fact that the software designers used scripts to determine the routing of the communication channels. Manual routing is found to be cumbersome and error prone, while adding little benefit when the task assignment is fixed. As the application was initially simultaneously designed and manually mapped to the RFDs, the mapping generated by the resource manager follows the same natural partitioning. In various experiments with faulty hardware components, this natural partitioning was abandoned and tasks were mapped and executed successfully on different RFDs then designed for.

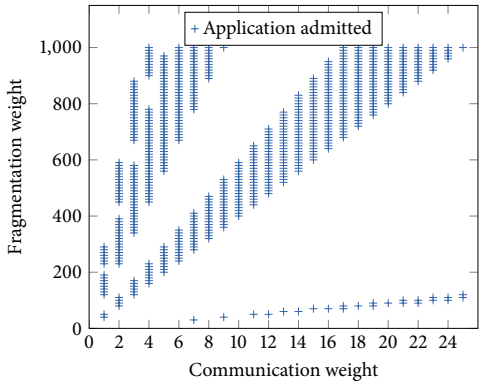


Figure 4.9: Admission of a beamforming application for various weights of the optimization objectives. Every point in $[0, 1, .., 25] \times [0, 10, .., 1000]$ is sampled.

As described in Chapter 3, the resource manager uses an optimization objective consisting of two factors; reducing resource fragmentation and minimizing the length of communication channels. The weight of each objective is made configurable, and different weights may result in different mappings. Figure 4.9 shows the mapping result with the beamforming application for various weights. A valid mapping

is only found for specific ratios between the fragmentation reduction objective and the communication minimization.

4.4.2 A SINGLE-FAULT SCENARIO

The flexibility of the run-time mapping system is evaluated by simulating a single hardware fault in one of the RFDs. In turn, one of the 100 NoC links, 16 routers, 9 Xentium® cores and 2 memory tiles is marked as faulty, after which we attempt to start the application.

The GNSS application

Table 4.2 shows the results of the 127 (100 + 16 + 9 + 2) different fault scenarios. The first column list the various types of resource used by the GNSS application. The second column gives the average utilization of the GNSS application per type of resource as opposed to the total availability on a single RFD. For example, the memory tiles have a utilization of 63%. The utilization of the NoC links cannot be defined or measured well, as the number of links used at a time changes over the lifetime of the application[2]. The third column shows that both of the memory tiles are critical; if one of them breaks down, the other memory tile lacks capacity to allow the GNSS application to start. Because only 8 out of 9 Xentium® cores are needed for the GNSS application (89% utilization), the setup tolerates one of the Xentium® cores to be faulty, so none of the Xentium® cores is critical. The fault tolerance measure in the last column specifies in what percentage of single-faults scenarios in the corresponding resource type the application is still able to start.

We also observed that the NoC is hardly used around the DM, and approaches maximum utilization around the memory tiles. This high variation in utilization of the interconnect indicates that some parts of it are more critical than other parts. Our experiment shows that 10 out of the 100 NoC links are essential to start the GNSS application. Some routers (5 out of 16) will render the RFD unusable for the GNSS application, as a faulty router will render multiple links unusable. The critical components in the RFD are shown in Figure 4.10 (marked with a red color). The memory tiles are heavily utilized, making each of them critical for the GNSS application. As a consequence, three out of the four access paths to these memory tiles are also found to be critical. MCP link number 2 takes the input stream from to the RF front end. MCP link number 5 leads to the GPD that manages the platform and performs part of the control of the application (see GPD and RFD 2 in Figure 4.4b). Thus, two off-chip links are essential for the application as well.

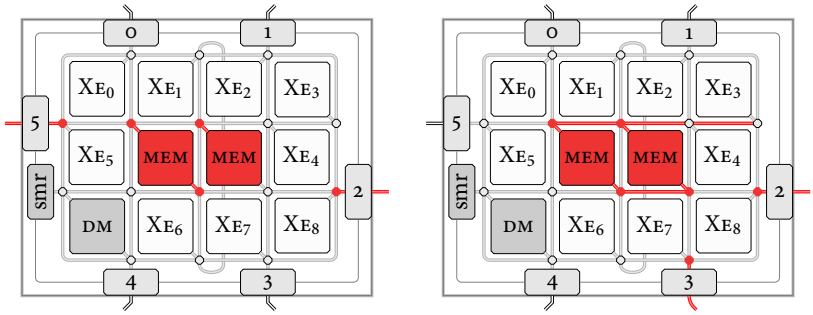
Figure 4.11 compares the number of links that are allocated by the GNSS application in a fault free RFD to mappings that are generated in the case that a single Xentium® is faulty. It shows that the impact of a single faulty Xentium® is minor to the GNSS application.

[2] For example, communication routes are configured to allow Xentium® cores to fetch the programming code from the memory tiles.

Table 4.2: Fault tolerance of the GNSS application on a faulty RFD.

Resource type	Utilization	Critical	Fault tolerance
NoC links		15 out of 100	85 %
NoC routers	75 ± 12%	5 out of 16	69 %
Xentium® core	89%	0 out of 9	100 %
Memory tiles	63%	2 out of 2	0 %

*The plus-minus sign (±) denotes one standard deviation.



(a) GNSS on RFD2 (b) BEAM8 on RFD2

Figure 4.10: Critical components on a degraded RFD.

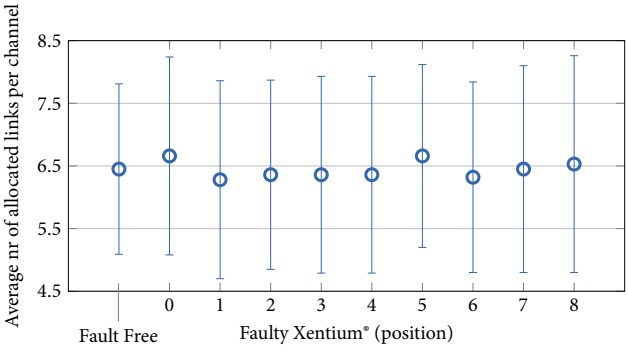


Figure 4.11: NoC utilization by the GNSS application on a fault free RFD compared to scenarios with a single faulty Xentium®.

The beamforming application

Similar to the GNSS application of the previous section, the fault tolerance of the beamforming application (BEAM8) is analyzed with RFD2 containing a single fault. Figure 4.10b shows the areas on the RFD that are critical to the resource allocation success of the beamforming application to the platform. This application uses the

FPGA both for input as well as for output streams. The communication paths back and forth between those ports and the on-chip memories are critical. Table 4.3 shows the observed fault tolerance of the beamforming application in single fault scenarios. Compared to the GNSS application, the BEAM8 application is slightly more sensitive to faults in the interconnect.

Table 4.3: Fault tolerance of the BEAM8 application on a faulty RFD.

Resource type	Utilization	Critical	Fault tolerance
NoC links		20 out of 100	80 %
NoC routers	94 ± 6%	6 out of 16	63 %
Xentium® cores	89%	0 out of 9	100 %
Memory tiles	100%	2 out of 2	0 %

* The plus-minus sign (±) denotes one standard deviation.

4.4.3 Multi-fault scenarios

Depending on the IC design, a single hardware fault may cause multiple components to become faulty. In this section, we evaluate scenarios where a hardware fault is not contained in a single component; we choose to evaluate the effect of a combined core, router and interconnect failure on the mapping result. All possible combinations of a single faulty Xentium®, a single faulty router and a single faulty link are considered. In this evaluation, the local links (back and forth) between the Xentium® and the first router are not considered; a fault on that link always makes the core inaccessible. While the multi-fault scenarios may include improbable faults, sufficient realistic scenarios are covered as well. If a fault combination prevents the application from running, each item in that combination is marked to be a critical resource. As the size of the three sets differ, some components may be tested more often than others.

Table 4.4 shows for each Xentium® and each router the ratio between the test cases that resulted in an admitted application versus the rejection of the application. The specific location of the Xentium® cores and routers in the RFD can be found in Figure 4.2. Due to the large number of NoC links that are tested (98), they are not listed in the results. For the Xentium® cores, in about 40% of the cases the GNSS application fails to run. The single fault scenario evaluation shows that none of the Xentium® cores is critical by itself, whereas some routers are critical for successful deployment of the GNSS application. That is observed in this scenario as well, where some of the routers are (close to) 100% criticality. Not all routers found to be critical in the single fault scenario are listed in Table 4.4 as 100% critical, as a specific combination of faults may trigger the mapping heuristic to consider a different part of the search space, resulting in a feasible mapping. So, for some cases in this scenario, the run-time mapper decided earlier in the mapping process to take resources from another RFD, while in the single fault scenario it got stuck in an infeasible search state. The results of Table 4.4 combined with the (unlisted)

NoC links are visualized in Figure 4.12a. The outcome is consistent with the single fault scenario of the previous section.

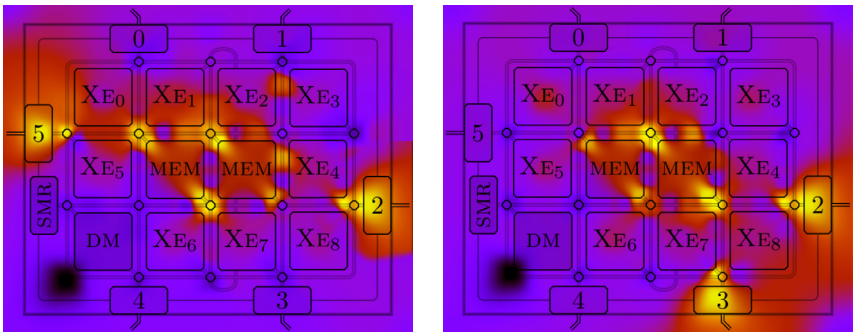
Table 4.4 and Figure 4.12b show similar results for the BEAM8 application. The connections to the FPGA (MCP 2 and 3) are single points of failure. Also, the communication resources around the highly utilized memory tiles are critical. The routers connected to the memory tiles closer to the FPGA are more critical, as they need to transport the input stream from the FPGA.

Table 4.4: Criticality of multiple hardware faults to the CRISP RFD.

Faulty element	GNSS application			BEAM8 application		
	Success	Failure	Criticiality	Success	Failure	Criticiality
Xentium 0	993	575	37%	821	747	48%
Xentium 1	833	735	47%	891	677	43%
Xentium 2	1001	567	36%	912	656	42%
Xentium 3	1000	568	36%	899	669	43%
Xentium 4	881	687	44%	936	632	40%
Xentium 5	976	592	38%	910	658	42%
Xentium 6	992	576	37%	892	676	43%
Xentium 7	969	599	38%	985	583	37%
Xentium 8	964	604	39%	918	650	41%
Router 0	699	183	21%	656	226	26%
Router 1	593	289	33%	589	293	33%
Router 2	737	145	16%	727	155	18%
Router 3	0	882	100%	744	138	16%
Router 4	40	842	95%	652	230	26%
Router 5	109	773	88%	0	882	100%
Router 6	645	237	27%	755	127	14%
Router 7	763	119	13%	744	138	16%
Router 8	703	179	20%	744	138	16%
Router 9	703	179	20%	745	137	16%
Router10	12	870	99%	333	549	62%
Router11	625	257	29%	0	882	100%
Router12	733	149	17%	0	882	100%
Router13	746	136	15%	745	137	16%
Router14	756	126	14%	730	152	17%
Router15	727	155	18%	0	882	100%

4.5 Conclusions

This chapter describes a many-core DSP platform developed in the context of the CRISP project. The demonstrator run-time mapping system is able to boot, configure and verify the entire platform using the built-in model of the platform. This alone already brings a huge benefit, considering the scale of the platform. This refrains developers from manual configuration and application mapping. Three



(a) The GNSS application.

(b) The BEAM (8B) application.

Figure 4.12: Criticality of combined Xentium®, router and interconnect faults on RFD2.

user applications and one test application were tested successfully with the run-time mapper to determine the resource assignment and configuration. Moreover, this chapter shows for these applications that the majority of the hardware faults can be circumvented. For both the GNSS and the beamforming applications, the interconnect is more critical than the Xentium® cores. Faults in the interconnect may render I/O ports and the highly utilized memory tiles inaccessible, refraining applications from execution.

The time required on the GPD to perform the required steps to start an application is higher than initially aimed for; up to three seconds for the largest application versus hundreds of milliseconds. However, the reader should take into account that the GPD is a low-end processor, and the given timing results are best interpreted relative to each other.

4.5.1 Outlook

To demonstrate the scalability of the CRISP concepts and the flexibility of the run-time mapping system, three GSP boards have been linked together to create a 146-core heterogeneous platform. Alternative configurations are very well possible, so that when required, other types of processing boards can be attached to this system as well. This setup is controlled in a centralized manner by the software stack described in this chapter, by only changing the platform model embedded in the run-time system. Within the research project STARS, described in the next chapter, we look into distributed and hierarchical approaches for controlling the platform, as the size of this platform and envisioned platforms is too large to be managed reasonably by a central entity (directly).

The run-time mapping system uses the algorithms of Chapter 3. That algorithm performs well when the resources requested, both the amount and location, match

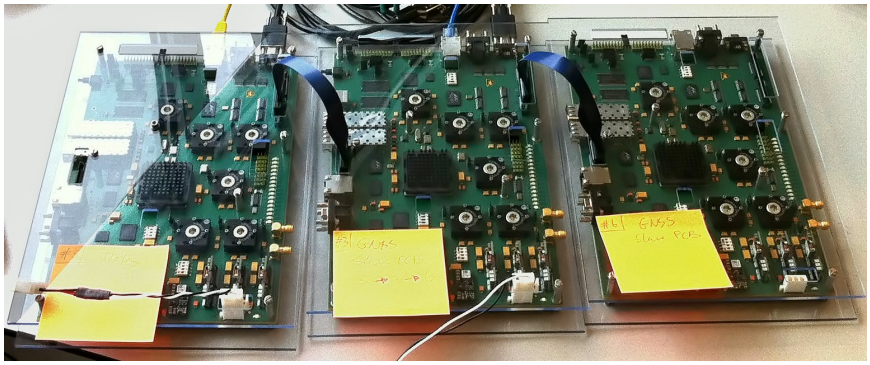
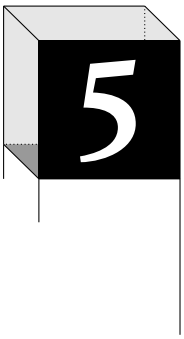


Figure 4.13: A GSP instance composed of three boards totaling 146 cores.

the resources available in the platform. In the CRISP case study, no false positives have been reported where the application is allowed to start on a platform using faulty or oversubscribed resources. On the opposite, false negatives are possible due to the lack of any back-tracking mechanisms in the algorithms. Due to the divide-and-conquer approach, decisions made early in the mapping process determine the options left for the remainder of the decisions to be made. In case of a mismatch between the resource request and resource availability, the quality of feedback information available in this approach is low. Especially in any non-centralized resource management approach, such information is required to (re-)negotiate on the distribution of resource requests over multiple controllers. In such a negotiation process, it may be beneficial to be able to continue with and/or 'repair' a partial resource allocation in an iterative manner when the difference between subsequent requests is limited. The algorithm of Chapter 3 is, with the requirements given before, less suitable for the STARS case study of the next chapter. Therefore, a different algorithmic approach is presented in Chapter 6.



Case Study 2: The STARS project

Abstract – *In the Sensor Technology Applied in Reconfigurable Systems (STARS) project reconfigurable systems are addressed using a hierarchical control system incorporating run-time mapping technology. The proposed system avoids the limitations of centralized management, at the cost of increased complexity of coordination. A multi-board demonstrator is built by extending the CRISP technology. A visualization framework of the actual resource availability and utilization gives insight into such a system. The need of the STARS project to improve upon the mapping technology of the CRISP project has been the main driver for the development of the guided local search approach of Chapter 6.*

Reconfigurability is an aspect well known to general purpose computing platforms. It denotes the capability of a system to change its behavior during run-time, in order to perform additional functionality, the same functionality in a more efficient manner, or different functionality due to changes in the environment or due to hardware faults. Reconfigurability in embedded systems is less common, as the design-time analysis of the possible configurations becomes too complex. Following up on the results of the CRISP project, the need for similar concepts applied to systems of increased scale is researched in this project. The Sensor Technology Applied in Reconfigurable Systems (STARS) project covers reconfigurability concepts ranging from reconfigurable antennas up to the the highest layers in the software stack.

Two problems arise when the concepts of Chapter 3 and Chapter 4 are applied to the STARS system. First, there is a limit to what amount of resources may reasonably be put under control of a single run-time mapper. At some point, it becomes infeasible to control the system from a single location, due to large latencies of operations to control the set of resources, or due to the limited communication bandwidth to the managing computing node. The second problem is that the STARS project proposes a layered system where the abstraction increases when going up in the layers. Decisions made at the top-most layer react on events in the environment of

STARS is partly funded by the Dutch fund for structural economical improvements, named Fonds Economische Structuurversterking (FES).

the system, resulting in a command to put the system in a configuration that corresponds with the functionality required. The type of application and corresponding parameters depend not only on the situation at hand, but also on the amount of resources available. This results in uncertainty on what functionality (application) needs to execute where and at what point in time, and if it even can be executed.

To maintain the effectiveness of the techniques proposed in this thesis, larger platforms should be separated into multiple domains, each having a controlling entity of its own. The domains are mostly defined by the architectural organization and natural boundaries of the components. The components of such distributed systems may cooperate in a way that resembles a market, in which sufficient information is available for decision-making. However, due to the added (communication) latency, the physical distribution of these domains impacts the expected performance negatively, while their logical distribution increases the complexity of the system. As a system becomes more complex, the opportunities for competition among these domains increases with a subsequent reduction in both performance and efficiency. Without any regulation, competition may defeat cooperation and the availability of relevant information may degrade. To introduce regulation in large systems like STARS, a *hierarchical control system* is proposed, which is a special form of a *networked control system*. The next section describes how a hierarchical system management is able to address both the problems of scalability and reconfigurability. The system we propose in the context of the STARS project is similar to the visionary statement of Brad Cox below:

> *Rather than thinking of 'command' and 'control' both operating from the top of the organization toward the bottom, we should think of command and control as an adaptive process in which 'command' is top-down guidance and 'control' is bottom-up feedback. Subordinates are guided not by detailed instructions and control measures but by their understanding of the requirements of the overall mission. The critical factor in such a system is to create command parameters and other systems features which provide the necessary guidance and level of understanding to create unity of effort without unnecessarily constraining the activities of subordinates. Complexity suggests it is rarely worth the effort trying to find the perfect plan or reach the perfect decision. It simply will not happen: there are too many interconnected variables. The single most important quality of effective command and control will be adaptability.*
>
> *– Brad Cox, 2004*

5.1 Hierarchical system management

Hierarchical control theory consists of two different foundations [42], i.e. multilevel control and multilayer control theory. In multilevel control theory, the control system is assumed to be decomposable in a set of coupled subsystems. The

decomposition can be done based on the physical or logical system structure. Each subsystem is connected to a local controller which solves a particular local control problem that is part of the overall problem. In order to achieve the overall control goal and to ensure consistent functioning of all subsystems, a higher level controller is designed to coordinate the local controllers by modifying their individual goals. In contrast to multilevel structures, multilayer hierarchical control theory is primarily concerned with different representations (models) of a system. In a multilayer structure the specification of control is split into algorithms which operate on different time scales and take their decisions based on models of different granularity. In this section, we propose a hierarchical control system for resource management that is both multilevel and multilayer. Figure 5.1 represents a *control node* of such a hierarchical system, consisting of a controller, a mapper and a deployer. The external interface of the control node allows for the creation of larger systems using a hierarchical composition of these nodes.

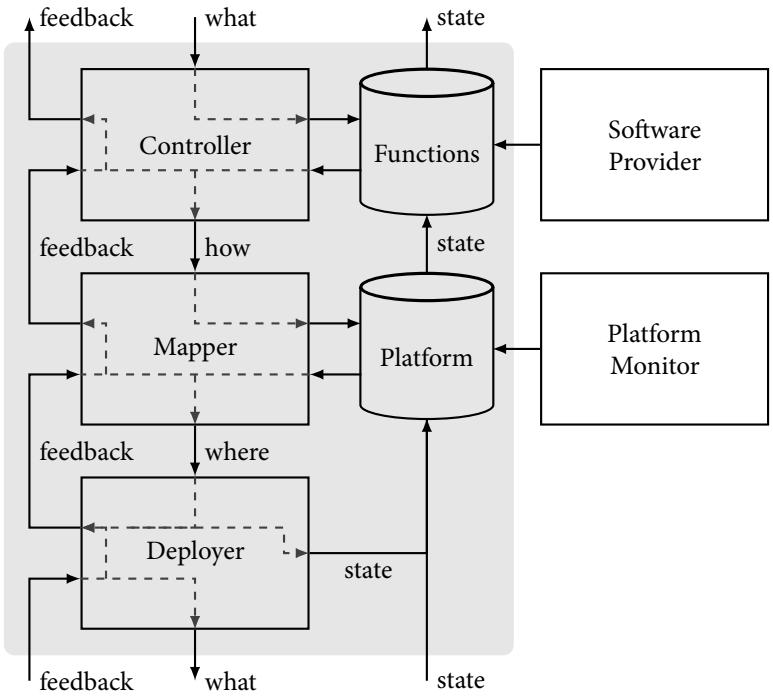


Figure 5.1: Hierarchical control structure for run-time mapping.

The top-down flow in Figure 5.1 *refines* coarse-grained requests the controller receives from the upper layers. The *what* as input to the controller describes the request of the outer shell of the functionality what needs to be performed. This may be the specification of the input and output streams with their specific location, as well as the function that needs to be performed inside the shell. The

controller attempts to make the translation from this high-level function description to a specific application graph with matching parameters and resource budgets. An information store provides the functions available at the current level of the system. To match the request of the outer shell, certain tasks (nodes) in the application graph are constrained such, that these tasks are pinned location-wise to input/output ports as specified by the upper layer. The output of the controller thus specifies *how* the requested functionality may be achieved. The mapper combines this information with the current status of the platform in order to allocate sufficient resources to the application. The resulting mapping describes *where* each constituent of the requested functionality has to be deployed. The deployer has platform specific knowledge to instantiate the application by configuring the platform accordingly, or by requesting more detailed functionality from a lower layer, which may be decomposed in multiple subsystems.

As a result of the request made by the upper layer at the controller, feedback information and state information is returned. A basic form of feedback is whether or not the requests made are successfully handled. A more advanced form of feedback, especially in the case where a request has been rejected, includes information on the resource scarcity and what part of the request can be handled. The full range of detail of state information cannot be returned, as the upper layer may not be able to interpret it due to lack of processing time or context information. *Abstraction* through data aggregation is used to represent the subtree of components. This gives each control node the perception that the system consists of two layers only, which is a common approximation technique [9, 82]:

> *Any hierarchical system of more than two levels can be approximated as a two-level hierarchical system. Hierarchical systems rooted at the third level may be aggregated as 'processors'. When system parameters and task characteristics are appropriately adjusted for the aggregated 'processors', results obtained in two-level systems can then be used to project overall performance of the system. Since the aggregated 'processors' are themselves hierarchical systems of fewer levels, a similar approximation can be applied to determine the corresponding system parameters [82].*

The coordination in a two-level hierarchical system may be implemented with iterative information transfer between the levels, solving a global optimization problem. Iterative coordination results in large delays, until optimal coordination results in the best achievable solution of the overall problem. A potentially faster approach is non-iterative coordination by means of a 'proposition-correction' protocol [110]. The upper layer sends an initial coordination to the sub-systems, which respond with a proposition for the problem posed. The upper layer then may correct for the diverted propositions and subsequently tell the subsystems to implement their proposition. This is the approach taken in the STARS project.

5.2 Demonstrator

A demonstrator platform is developed that consists of four CRISP boards and two TI TMDS EVM 6678L evaluation boards, each hosting a TI TMS320C67x 8-core DSP. Figure 5.2 shows the evaluation board, whereas Figure 5.8a shows a visualization of the demonstrator system at the board level. The control software is configured as a two layer hierarchical system, where the CRISP boards are equipped with the run-time mapping system of Chapter 4. The four CRISP boards are controlled by one of the TI boards. The TI board contains a run-time mapping system different from the CRISP system. Both implementations conform to the hierarchical control structure of Figure 5.1. The second TI board is used for data generation and injection into the demonstrator, emulating a phased antenna array.

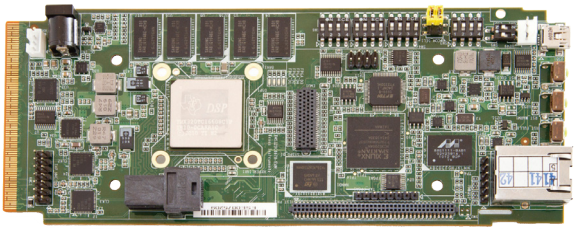


Figure 5.2: TI TMDS EVM 6678L evaluation board.

The beamformer application developed in the CRISP project has been adapted for use in the STARS demonstrator. The CRISP boards are used to perform the first stages of digital beamforming. After data reduction in the first beamforming stage , the signals of all the CRISP boards are combined on one of the TI boards to perform the remaining stages of the beamforming. This demonstrator represents a phased antenna array with up to 64 channels of input, distributed to the available CRISP board. The user interface of Figure 5.4 is used to dynamically generate the input data, depending on the location and frequency range of the simulated targets and depending on the amount of noise applied. The system is capable of outputting up to 20 beams; the output needs to vary between applications. The generic beamformer system, as shown in Figure 5.3 is used to service multiple applications, of which two are described in more detail:

Radar surveillance: detects targets with 8 beams at a normal level of detail (number of range samples). It uses a wide beam (azimuth coverage) per scan.

Radar classification: classifies targets with one beam at a high level of detail and a narrow beam to focus on the target.

Each of these applications has configurable modes, allowing for some degraded performance when insufficient resources are available for the full performance mode. This allows for various combinations of application modes running alongside each other. The user interface shown in Figure 5.5 provides control over the active set

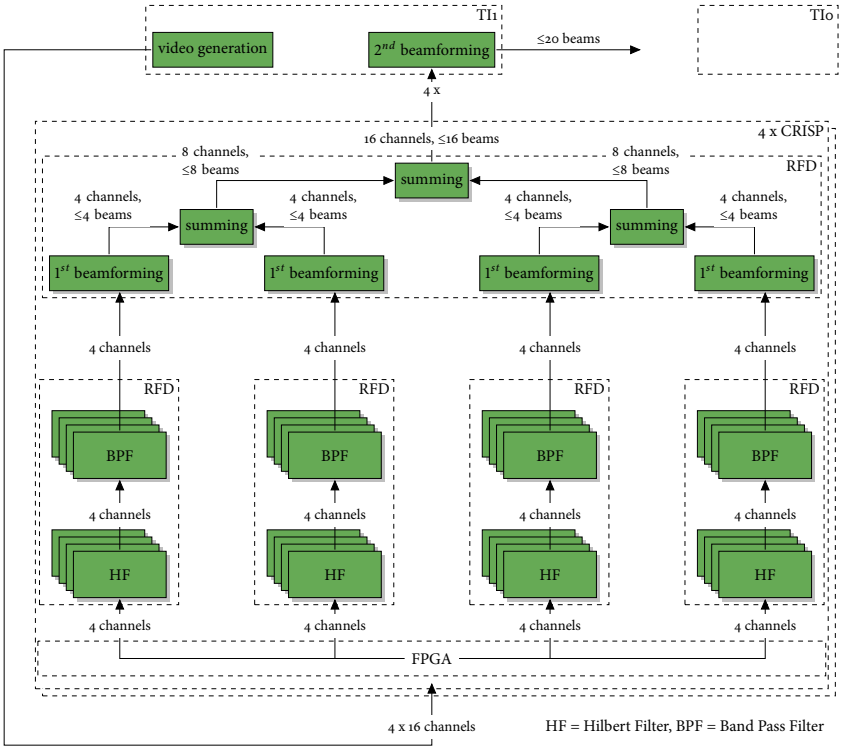


Figure 5.3: STARS demonstrator application.

of applications running on the system. The left panel (light grey) displays an entry for the control software itself, and for each of the three applications. It shows the requested application mode, including an inactive mode, and the actual mode being executed. The actual mode of an application may be of a lower quality of service (QoS) level due to a lack of resources. The right panel (dark gray) shows the 6 boards of the demonstrator system, with a representation of the mapping of the active application set to the boards. The two layers of the control system are presented as such. The control node on the highest layer of the hierarchical system (the TI1 board) performs coarse-grained mapping of functionality across the available boards, using a first fit allocation algorithm. At the lower layer (the CRISP boards), the run-time mapping system of Chapter 4 is responsible for the fine-grained resource management.

Results

The STARS demonstrator has been used in multiple occasions to provide proofs of concept of maintaining a higher level of flexibility in large scale embedded systems such as STARS, as opposed to the more conventional (static and inflexible) way

of mapping applications onto platforms. For the set of applications described in this chapter, the system functioned correctly. In Figure 5.6a, typical beam patterns are observed as output of the surveillance application. After surveillance with low detail, the beams that contain (possible) targets can be scanned with higher detail. An example output of such a classification application is shown in Figure 5.6b. Note that the output of Figure 5.6b is not corresponding to Figure 5.6a due to limitations of the demonstrator software.

In the demonstrator, the dominant factor in the time required to make changes to the active application set is attributed to the TCP/IP communication stack used. As the performance of the control nodes was second to the functionality and presentation of the demonstrator, reliable quantitative measurements are not available.

While Figure 5.5 gives an overview of the status of the entire platform, there is a need to monitor and visualize the status of the platform in more detail, both for demonstration and research purposes. Specifically, the actual resource usage and the location of the resources in use (by applications) is insufficiently presented in Figure 5.5. The next section describes a visualization technique and framework that improves on this aspect.

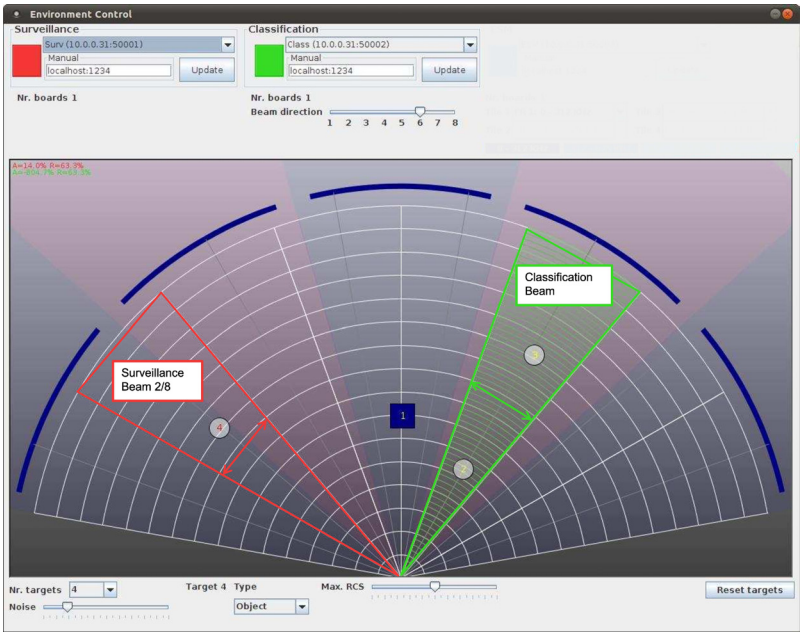


Figure 5.4: Environmental control interface of the STARS demonstrator.

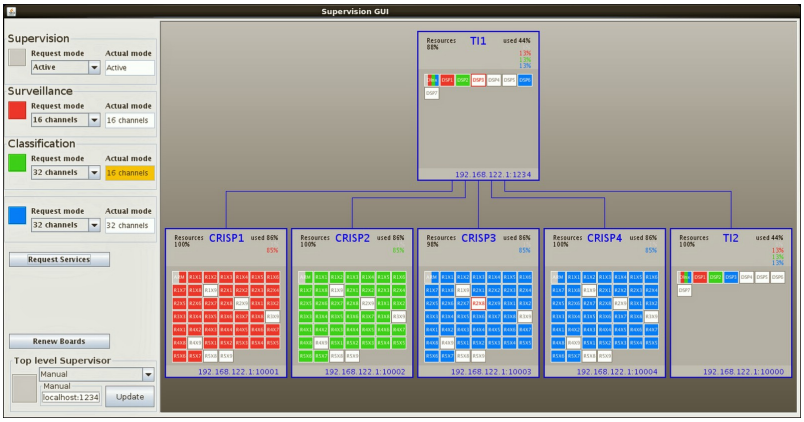
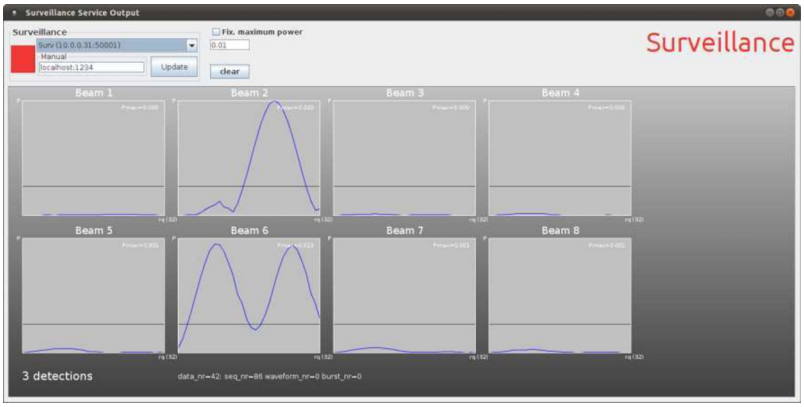
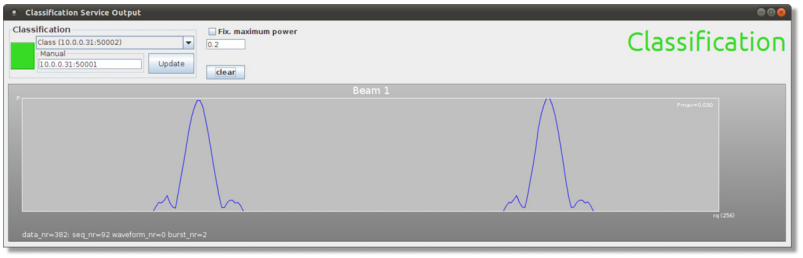


Figure 5.5: Application control interface of the STARS demonstrator.



(a) Output of the surveillance application.



(b) Output of the classification application.

Figure 5.6: Example output beam patterns of the STARS demonstrator applications.

5.2.1 VISUALIZATION

The inherent complexity of large-scale heterogeneous and distributed systems may result in a massive amount of status information, often too large to be interpreted effectively by humans. Visualization of this information is used for debugging and monitoring such systems. To the best of our knowledge, no generic tool exists for visualizing networks and (computational) resources simultaneously. Among the weaknesses in existing tools that provide the required functionality are:

- manual (static) layout of the visualized platform,
- lack of interactivity,
- scalability issues with large networks,
- specialized software requirements and
- limited visual display of quantitative information.

To improve each of the weaknesses just described, we extend a visualization technique for network links from a publicly available prototype. The 'NetMapJs' prototype [88] introduces a new way of displaying quantitative data within the edges of a network graph. As the main focus in [88] is on network links, the nodes of the network do not present any quantitative data. Figure 5.7 gives an annotated view of how the utilization (bandwidth) of a network connection is visualized. The design uses line thickness to show the link capacity and color for utilization, as this is the most accurate method for encoding quantitative variables [105]. Colour is effective at displaying categorical variables which is why it is being used in the visualization.

In our case, both the network and the (processing) elements must be considered, as formalized in Chapter 2. An element may provide one or more resources, with limited capacity. We extend the visualization of the network with hierarchical elements, where elements are represented with a segmented donut chart, with a segment for each resource type, as is shown in Figure 5.7 as circles to the left and right of the figure. Each segment is filled with a colour to indicate the utilization of the resource, where a completely filled segment corresponds with full utilization. Similar to the network links, the colour of the filling may be used to display a categorical variable, such as the health status of the resource. Composite elements, such as the right circle in Figure 5.7 may aggregate all the resources with their allocated and available capacities of its children. The inside of the circle representing a composite element may contain nodes that represent the elements at lower layers. This allows a user to quickly obtain an overview of a large-scale system.

Automatic layout

A visualization should provide a good layout of the nodes onto the canvas, minimizing clutter and overlap. Manually positioning the nodes potentially gives good results (see Figure B.5a), because the resulting layout may approach the physical or logical structure of the platform itself. On the other hand, manually defining a

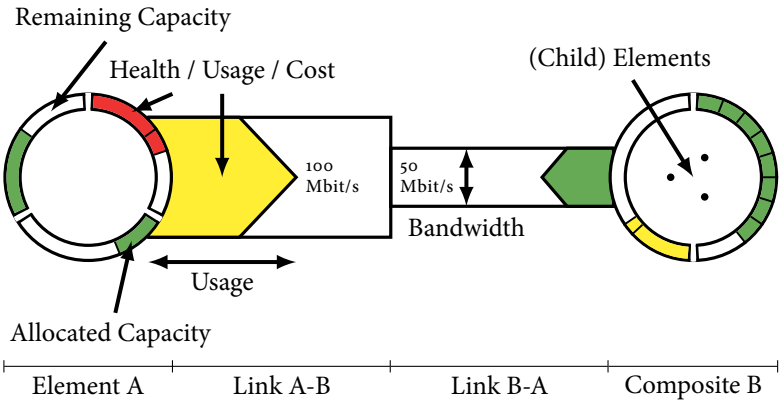


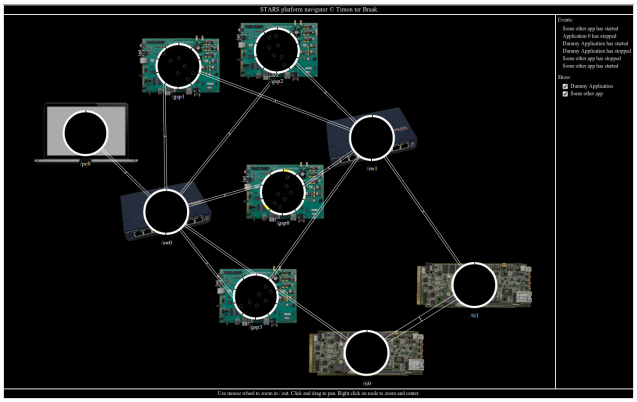
Figure 5.7: Visualization of resource usage in a hierarchical system.

layout can be too time consuming for large-scale graphs, or inconvenient for (platform) graphs that change frequently. Generating a layout of the network graph automatically, as shown in Figure B.5b, allows the visualization tool to process arbitrary graphs, thus, extending the applicability of the visualization tool. As force-directed algorithms are a common choice for automatic layout of graphs, our visualization framework employs the *ForceAtlas2* [66] algorithm, which can be configured to exhibit a good combination of speed and quality.

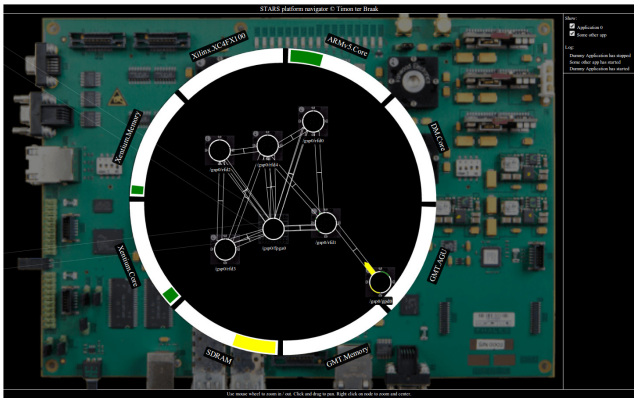
Semantic zooming

Layout algorithms can be good for displaying large graphs clearly but there is a limit to their effectiveness. At some level of network size or density the limits of these algorithms will be reached and good layouts will no longer be guaranteed. It is then likely that a user can no longer extract information and patterns from such large networks. Combining layout algorithms with other techniques may be useful to maintain scalability [51]. One way this can be done is through data aggregation and filtering to reduce the amount of information presented in the view.

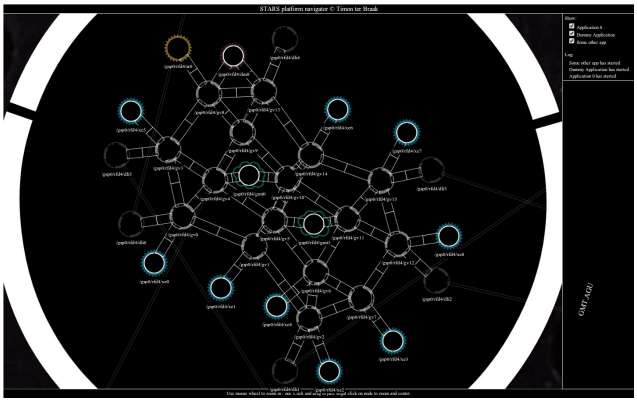
Another technique for information reduction is semantic zooming [89]. This is different from geometric zooming in that changes are made to the level of information displayed while moving towards a specific area of a graph. NetMapJs [88] introduced semantic zooming along with panning to effectively navigate throughout large networks. We also applied this technique to the composite elements in Figure 5.7. Using the STARS demonstrator as example, Figure 5.8 shows three zoom levels, with the system-level view at the top. Zooming in provides the user with a view on one of the processing boards, while further zooming shows the internal state of a multi-core processing device. The entire platform consists of around 587 nodes, and 1870 edges, which is too large to display entirely and show meaningful information at the same time.



(a) System-level view (STARS demonstrator).



(b) Board-level view (single CRISP board).



(c) Chip-level view (single RFD).

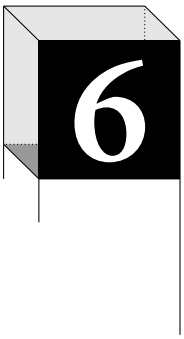
Figure 5.8: STARS platform navigator at various zoom levels.

5.3 Conclusions

For large-scale platforms, a hierarchical control system is proposed with a node structure consisting of a controller, mapper and deployer. A two layer system is demonstrated successfully with three applications developed in the STARS project. The visualization framework proposed is able to dynamically navigate through the hierarchical system, and to display both detailed as well as aggregated data on the real-time resource usage.

5.3.1 Outlook

As opposed to the STARS demonstrator, where the design of the control nodes differs between the layers of the hierarchical system, a single implementation is desired that can be used at all system layers. This would avoid additional development effort when the hardware configuration of the system under control is changing, as a single run-time mapping system can then be deployed to any control node. Moreover, the algorithm of Chapter 3 has some limitations (as discussion in Chapter 4) that makes it less suitable for the hierarchical control system proposed and demonstrated in this chapter. The STARS case study is, therefore, one of the drivers for further research and development of the run-time mapping technology of Chapter 3. Aiming for uniformity of the run-time mapping system, a different approach using meta-heuristics is described in the next chapter. As the development of these meta-heuristics is done simultaneously with the embedding of CRISP technology into a STARS demonstrator of increased scale, integration of the meta-heuristical approach of the next chapter into the STARS demonstrator proved not to be feasible within the scope of the STARS project.



Metaheuristics: Guided Local Search

Abstract – *The work presented in this chapter builds upon state-of-the-art work of Yagiura et al. [122, 123, 125] on metaheuristics for various generalizations of the generalized assignment problem. Specifically, we focus on the guided local search (GLS) approach for the multi-resource quadratic assignment problem. The quadratic assignment problem defines a cost relation between tasks and between elements. We generalize the multi-resource quadratic assignment problem with the addition of an interconnect with limited capacity and a communication topology between tasks. Numerical experiments show that the performance of the approach is comparable with industry standard integer linear programming solvers. The footprint, the time versus quality trade-off and available metadata make guided local search a suitable candidate for run-time mapping.*

The complexity of the multi-resource quadratic assignment and routing problem (MRQARP) renders exhaustive methods to find the optimal solution inapplicable. Instead, we may accept suboptimal solutions and aim for short computation times and low memory usage. More precise requirements are not available because the approach should be suitable for a wide range of systems and applications. Even within the same operational context, the non-functional requirements may change over time and between resource requests. As a consequence, it is not feasible to establish precise requirements. The heuristics presented in Chapter 3 already trade optimality of the solution for feasibility and a reduction in computational effort. In the context of real-time systems, we additionally require that a solution is found within a certain duration, which may not even be known on beforehand. Then, a solution is of no use when it arrives after the event marking the end of the allowed duration. The guided local search algorithm described in this chapter is an *anytime algorithm*; i.e. it provides increasingly better solutions when additional computation time is allowed. This means that at any time the algorithm is stopped, the best known solution available at that time can be obtained. This is

The guided local search technique proposed in this chapter is published in [TDtB:7].

one of the properties that makes GLS an interesting technique to enable run-time mapping. The GLS approach described in this chapter is based on related work that targets the MRQAP [125]. We tailor their approach to our problem domain, and extend it with communication routing.

Some terminology is introduced first, which is used in the remainder of this chapter. The GLS technique is explained initially for the task assignment problem only. This chapter gradually builds up a complete framework, starting with the basics of local search. Thereafter, we describe the coordinating process controlling the local search (the guidance part), followed by a mechanism that adds additional robustness to the search technique. Full understanding of the approach for the task assignment enables us then to incorporate the communication routing subproblem into the search technique. The chapter ends with the numerical results obtained through an evaluation of our approach on an extensive dataset.

Terminology A problem instance has a search space $\mathcal{U}$, in which each point represents a *solution*. The set $\mathcal{U}$ is ordered by an objective function $f : \mathcal{U} \to \mathbb{R}$. A feasible region covers the points in the search space for which all constraints are satisfied. A search space contains zero or more feasible regions. We call a point within a feasible region a *feasible solution*, and points outside the feasible region *infeasible solutions*. The set of all feasible solutions for a given problem instance is represented by set $\mathcal{F} \subseteq \mathcal{U}$.

For any solution $\sigma \in \mathcal{U}$, the *neighborhood* $\mathcal{N}(\sigma)$ consists of solutions which are topologically 'close' to σ. For at least one solution $\bar{\sigma} \in \mathcal{N}(\sigma)$, it holds that

$$f(\bar{\sigma}) \leq \min_{\sigma' \in \mathcal{N}(\sigma)} f(\sigma'). \tag{6.1}$$

Such a solution $\bar{\sigma}$ is a *locally optimal solution*, and this solution is not necessarily feasible. Within the search process, the current best known feasible solution is referred to as the *incumbent solution* $\hat{\sigma} \in \mathcal{F}$. In case of a minimization problem, it holds for a globally optimal solution $\sigma^* \in \mathcal{F}$ that

$$f(\sigma^*) \leq \min_{\sigma \in \mathcal{F}} f(\sigma). \tag{6.2}$$

Instances of the MRQARP defined in Section 2.1.4 are referred to a problem Z. The objective value of Z is given by $Z(\sigma)$, and a lower and upper bound of this value is given by $\check{Z}$ and $\hat{Z}$ respectively. Figure 6.1 illustrates the terminology used (consistent with [33]) throughout the remainder of this chapter.

6.1 Guided local search

In this section, we describe a metaheurstic to solve the MRQARP. A metaheuristic is an iterative search method that internally uses a *local search algorithm*. Local search is a procedure that alters a solution σ slightly, hoping to get an improved solution σ' in the neighborhood $\mathcal{N}(\sigma)$. As the modifications made to a solution

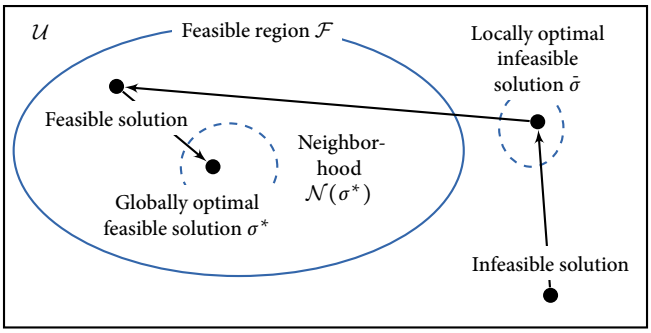


Figure 6.1: Terminology used throughout this chapter regarding search space and solutions.

are typically small, local search methods have a tendency to keep cycling around in the same neighborhood, unable to get out. The purpose of the metaheuristic on top of the local search is to change the behavior of the local search algorithm in between iterations in order to steer the search out of these potential suboptimal, or even infeasible regions.

The metaheuristic known as guided local search (GLS) is considered to be a special case of tabu search [38]. Tabu search uses memory structures to form a tabu list, which contains previously discovered solutions or problem features that are no longer allowed to be part of solutions to be discovered next. GLS penalizes problem features that should be avoided in the next iteration. Therefore, the objective function of Z is augmented with these penalties in order to reshape the search space. Figure 6.2 plots an example objective value of problem Z, indicated by the blue line. The black line illustrates the path followed by the local search procedure. Initially, the local search procedure moves from an infeasible solution σ (below the function Z) to a local optimal solution $\bar{\sigma}$. To escape the local minimum, the penalties in the objective function are changed, resulting in penalized objective functions as indicated by the dashed lines. As a result, the local search procedure is moving away from a previously discovered local minima to different parts of the search space.

Figure 6.3 shows the basic operations in such a GLS framework. A single iteration starts with the generation of a solution σ. This solution is likely not globally optimal, and probably not locally optimal, or not even feasible, depending on the solution generation mechanism. A local search procedure is then applied to solution σ, resulting in a locally optimal solution $\bar{\sigma}$. As a last step, solution $\bar{\sigma}$ is used to update the penalties used by the local search procedure. The updated penalties ensure that bad decisions made in the current iteration, become more expensive in the next iteration. This concludes a single iteration of the algorithm.

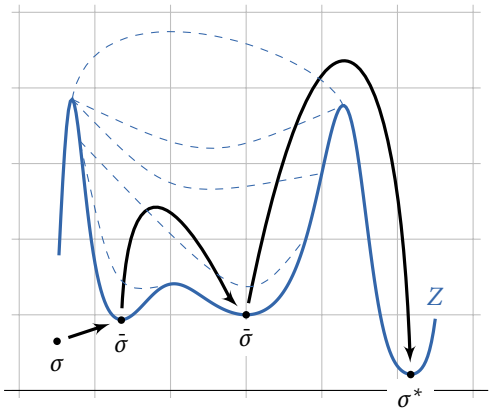


Figure 6.2: Penalties adjust the objective function of problem Z to steer the search out of local optima.

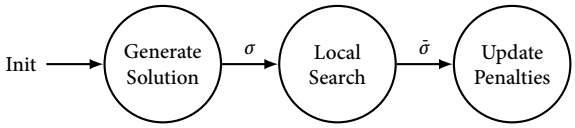


Figure 6.3: Basic operations of a single iteration in a guided local search framework.

In the remainder of this section each of the three steps is described in more detail, and the framework as a whole is refined with additional mechanisms to control the search process.

6.1.1 Initial solutions

The purpose of the local search procedure in Figure 6.3 is to make a solution σ locally optimal, resulting in a solution $\tilde{\sigma}$. An initial set S of solutions is generated to provide a solution $\sigma \in S$ as input to the local search procedure. Assuming that no additional information is known on the search space, it is beneficial to have distinct solutions in S, such that the search space is well represented. The solutions in S may be generated randomly, or may be constructed using other heuristics or algorithms. Alternatively, the set of initial solutions may be determined at design-time [104]. GLS allows for such a *hybrid mapping strategy*, where precomputed solutions are used to seed the local search process. These precomputed configurations may need to be 'repaired' if the circumstances at run-time are different from the ones analyzed at design-time. Repairing a known good solution is likely to require less effort than computing a solution from scratch. The run-time overhead of this hybrid mapping strategy is then vastly reduced while maintaining all the flexibility to adapt

or optimize the precomputed solution. Evaluation of this approach is considered as future work.

In our run-time mapping approach, we combine two methods to obtain a set of initial solutions. Randomly generated solutions provide a good distribution of starting points over the search space. These solutions are probably of poor quality, and relatively much effort is required to improve them through the local search procedure. In this improvement procedure, it is beneficial to have some knowledge on what problem features are good and should be maintained, and what problem features are disadvantageous. This knowledge is then used to initialize the 'tabu list'. To this end, a Lagrangian relaxation technique is used to obtain a relatively good, but potential infeasible solution. Lagrangian relaxation is a well known technique [32], and will be explained next in detail for the MRGAP.

Lagrangian relaxation

Referring back to Section 2.1.5, the run-time mapping problem is made difficult by the capacity constraints. The problem becomes much simpler when these constraints are removed. Complete removal of these constraints gives hardly any valuable information, because the structure of the original problem is almost completely lost. *Lagrangian relaxation* is a technique that alters a problem such that difficult constraints are removed from the problem, and are represented in the optimization objective instead. In the resulting *Lagrangian dual* problem, the constraints may be violated at the cost of the objective value. A solution to the Lagrangian dual is a lower bound (in case of minimization problems) for the original problem, which we refer to as the *primal problem*. Hence, both the lower bound and the corresponding solution provide valuable information we may exploit while solving the original problem.

A natural relaxation [32] for the task assignment problem is obtained by removing the 'hard constraints' of Equation 2.21, which model the limited resource capacity of elements. The constraints are replaced by a vector of *Lagrangian multipliers* λ in the objective function of Equation 6.3 to penalize any oversubscribing of resources.

$$Z_D(\lambda) = \min \quad \sum_{e \in E} \sum_{t \in T} (\text{cost}(t,e) x_{te} + \sum_{k \in R} \lambda_k (x_{te} r_{te}^k - c_e^k)), \tag{6.3}$$

$$\text{s.t.} \sum_{e \in E} \sum_{t \in T} x_{te} = 1, \tag{6.4}$$

$$x_{te} \in \{0,1\}, t \in T, e \in E.$$

The Lagrangian dual problem Z_D is a convex minimization problem [94], which can be solved efficiently. The *subgradient method* [50] is a simple iterative algorithm to minimize nondifferentiable convex functions. With an initial vector λ having all ones, the dual problem is solved by assigning each task to the element with the least penalized cost. The resulting solution gives a weak *lower bound* on the

objective function of the original problem Z. It is a lower bound due to potential severe violation of the dualized constraints. In an attempt to improve this bound, the Lagrangian multipliers are adapted before starting the next iteration.

Subgradient
The derivative of a one-dimensional function can be generalized to the *gradient* of a function in multiple dimensions. These concepts can be further generalized to non-differentiable functions. The *subgradient* is the 'derivative' of a non-differentiable function with multiple variables.

One iteration i of the subgradient method consists of taking a step size $s^{(i)}$ along the negative direction of the subgradient

$$g_k^{(i)}(e) = \sum_{t \in T} (x_{te}^{(i)} r_{te}^k) - c_e^k, \quad \forall k \in R. \tag{6.5}$$

of the objective function at the current point $e \in E$. The negative direction is taken because we deal with a minimization problem. With certain sequences of step size $s_k^{(i)}$ the iteration process converges [2] to a local minimum. A widely used step size [32] is

$$s_k^{(i)} = \frac{\hat{Z} - Z_D(\lambda_k^{(i)})}{\|g_k^{(i)}\|_2^2}, \quad \forall k \in R, \tag{6.6}$$

where $\hat{Z}$ is an upper bound on Z_D; $\hat{Z}$ is the value of the incumbent solution $\hat{\sigma}$ or a large value in case no such solution is available yet. The squared Euclidean distance $\|g_k^{(i)}\|_2^2$ sums per resource type k over all elements $e \in E$ the difference in the amount of available resources and allocated resources k[1]. We smoothen this metric by using a filtered subgradient [15], where we combine the knowledge of two iterations. The denominator of Equation 6.6 is then replaced by:

$$\|0.75 g_k^{(i)} + 0.25 g_k^{(i-1)}\|_2^2.$$

The step size $s_k^{(i)}$ thus divides the gap between the upper and lower bound on Z by the degree of oversubscription in the current assignment. This results in a greater step size when the gap is large and the oversubscription is minor, as well in a smaller step size when the gap is small and the oversubscription extensive. With each iteration, the Lagrangian multipliers λ_k are adjusted in order to converge to feasibility of the original problem, using

$$\lambda_k^{(i+1)} = \lambda_k^{(i)} \cdot s_k^{(i)}. \tag{6.7}$$

[1]The p-norm of a vector is defined as $\|x\|_p = (\sum_i x_i^p)^{\frac{1}{p}}$. Here, the 2-norm squared results in $\|x\|_2^2 = x_1^2 + x_2^2 + \ldots + x_n^2$.

The solution to the dual problem Z_D changes when the multipliers change sufficiently. If the best known lower bound Z_D is improved by such a solution, the corresponding assignment is stored. It is very difficult to determine a good termination criterion for the subgradient optimization procedure [32]. Therefore, we allow a bounded number of iterations without any improvements; in this work, the limit is set to 5 iterations, after which the process is terminated. The solution of Z_D is added to solution set S.

Quality estimation In this thesis, we define the *duality gap* as the difference between the value of any dual solution (Z_D) and the value of a feasible primal solution (Z). The duality gap then gives a quality estimation of a solution for the primal problem Z. This gap becomes smaller when a new incumbent solution is found (lowering the upper bound), or when an improved dual solution is found (raising the lower bound of the primal solution Z). This information is used later in this chapter, specifically in Section 6.1.3.

As the result of the first step of Figure 6.3, a set S of solutions is generated. A local search procedure is applied to each solution $\sigma \in S$ in order to get a set of locally optimal solutions. The local search technique is described next.

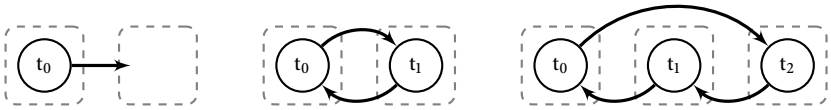
6.1.2 Local search

Local search is a technique that starts with a solution $\sigma \in S$, where S is a set of solutions obtained by the procedure described in the previous section. Through alternations on solutions, called *moves*, the search iteratively traverses the search space towards improved solutions. A move defines the set of possible solutions that can be obtained by changing solution σ; the set of reachable solutions is then defined as the neighborhood $\mathcal{N}(\sigma)$ of a solution σ. In this work, three moves are defined together with their corresponding neighborhood. More complex moves that extend the local search into larger neighborhoods may be required to find better solutions or may be required to solve certain problem instances at all. In the context of this work, the added computational demand is not easily justified. Moreover, the evaluation at the end of this chapter shows the strength of the moves that are used in the local search:

Shift move:
a shift move reassigns a single task to another element [79].

Swap move:
a swap move exchanges the assignment between two tasks [57, 79].

Chained shift move:
a chained shift move removes (ejects) a task t_0 from its assigned element e_x. Then another task t_1 is shifted from element e_y to element e_x, increasing the availability of resources on element e_y. Recursively, a task is shifted from element e_z to element e_y. As a last step, the chain is completed by assigning task t_0 to element e_z. This procedure is based on ejection chains [122, 123].



(a) Shift (b) Swap (c) Chained Shift

Figure 6.4: Moves used in local search

Local search is an anytime algorithm; it returns a solution at any time the search is terminated. The termination criterion can be based on execution time, a fixed number of iterations, or when the solution can no longer be improved. For any neighborhood structure, a rule needs to be defined to update the current solution when improvements are available. One rule is to select the first improvement found, and another rule is to select the best possible improvement. When such an improvement is selected in the current neighborhood, a new neighborhood is constructed. Alternatively, we adopt Heider's rule [33] which defines that the first possible improvement found needs to be chosen, after which the search is *continued* in the current neighborhood. This improves the locality of the search procedure, while it reduces computation time for the construction of neighborhoods. Ordered on the complexity of the operation, we first search in $\mathcal{N}_{shift}$, followed by $\mathcal{N}_{swap}$ and lastly in $\mathcal{N}_{chain}$. When an improved solution σ' has been found, the local search is continued by searching in $\mathcal{N}_{shift}(\sigma')$, until no improvements have been found in the entire neighborhood $\mathcal{N}$, where $\mathcal{N}$ is defined as

$$\mathcal{N} = \{\mathcal{N}_{shift} \cup \mathcal{N}_{swap} \cup \mathcal{N}_{chain}\}. \tag{6.8}$$

The lack of improvement moves then determines the termination of the local search procedure. Figure 6.5 shows how the local search procedure fits in the overall framework.

Efficient implementation

The objective function of Z is used to judge whether a move improves the solution. Because a move only changes a small part of the solution, evaluation of the objective function over the entire solution is quite costly as it recalculates a lot of values that are not impacted by the move. Therefore, we calculate and store the deltas d of the cost function for every possible move. As such, we can determine the impact of a proposed move without actually performing it. On top of that, we employ memoization of the deltas, and keep them updated after a move has been performed. The improvement evaluation function does consider penalty weights, which will be described in the next section. The entire set of memoized deltas only have to be recalculated when the penalty weights have changed. More detail on this optimization can be found in [125].

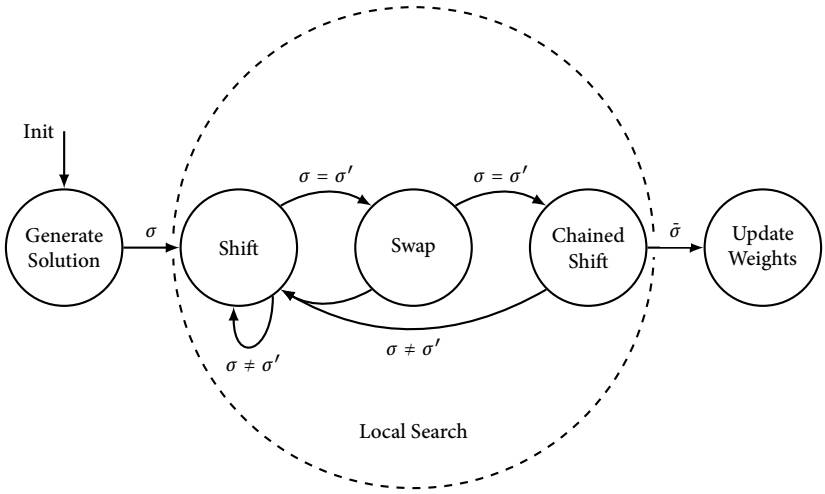


Figure 6.5: Guided local search framework refined with the local search procedure.

6.1.3 GUIDANCE WITH PENALTY WEIGHTS

A solution is infeasible when some of the resources are oversubscribed. To improve the feasibility of a solution, moves have to be performed that trade solution cost for feasibility. Concretely, this means that some tasks may need to be mapped to less preferred elements to adhere to the capacity constraints. To steer this process, the cost function penalizes the extent to which a resource is oversubscribed. Per resource k at element e, the contribution of task t to the oversubscription is penalized with a factor of p_e^k. Equation 6.9 defines the penalized cost function, which adds a penalty to the normal cost function in case of any oversubscription. To calculate the contribution of a task t to the oversubscription, we subtract the amount of oversubscription of that resource without task t. In case of any contribution to the oversubscription (the resulting amount being positive), the degree of oversubscription is multiplied with the penalty imposed on that specific resource.

$$pcost(t,e) = cost(t,e) + \sum_{k \in R} p_e^k \times max(0, ((\sum_{t \in T} r_{te}^k x_{te} - c_e^k) - (\sum_{t' \in T \setminus \{t\}} r_{t'e}^k x_{t'e} - c_e^k))) \tag{6.9}$$

The penalty weights have to be initialized with a certain value without a priori knowledge of the resource scarcity. A simple but effective approach is to initialize all penalties with the value 1.0. As a result, oversubscription of resources is penalized proportionally with the contribution of the task to the oversubscription. Such penalties merely take the resource capacity limitations into account. Substantial resource oversubscription may occur if certain resources are more favorable than others. In those cases it might be possible that the penalty given for oversubscription no longer outweighs the increased cost of moving to a less desirable element.

An alternative method to initialize the penalty weight considers, in addition to the resource capacities, the desirability as expressed in the cost function of a task for one element over another. The aim is to balance the average increase in the cost of a move towards a less desirable element with the average decrease in the penalty of using oversubscribed resources. The problem is that a single penalty is set for a single resource, whereas the desirability for that resource can be expressed for every task in the application. This problem is resolved, per resource k of an element e, by fitting a straight line through the cost coefficients $cost(t, e)$ of all tasks $t \in T$, as a function of the amount of resource r_{te}^k required. This relationship between cost and relative resource scarcity is a *simple linear regression*. Fitting a linear model through the data points is solved by a set of normal equations derived from a linear least squares problem [122]. We then choose weights that minimize the residuals of the difference between the cost of assigning tasks to particular elements. The Gauss-Seidel iterative method is used to solve the resulting linear system. The next paragraph describes this method in greater detail, after which we evaluate both initialization methods.

The Gauss-Seidel method for penalty weight initialization Per resource type k, the total resource demand $\sum_{t \in T} r_{te}^k$ of tasks at each element $e \in E$ is specified in matrix A, and the associated cost in vector b (as defined by the cost function $cost : T \rightarrow E$). The resource demand of A apparently has an unknown value x that together add up to b:

$$Ax = b \tag{6.10}$$

This linear system is typically overdetermined and inconsistent[2]. Therefore, we approximate the vector of unknowns x by solving the equivalent linear system

$$A^T Ax = A^T b \tag{6.11}$$

called the normal equations. The normal equations minimize the sum of the square differences between the left and right hand size of the equation. The Gauss-Seidel iterative method solves the left hand side of the linear system for x. In a single iteration step, the unknowns x at iteration i are determined by the values from the previous iteration $i - 1$, and any values that are already computed in the current iteration[3].

$$x_e^{(i)} = \frac{b_e - \sum_{e'<e} a_{ee'} x_{e'}^{(i)} - \sum_{e'>e} a_{ee'} x_{e'}^{(i-1)}}{a_{ee}} \tag{6.12}$$

The least-squares solution x reflects the relative value of a resource, which indirectly gives a task's desirability for one element over another. Low values for x correspond

[2] An overdetermined linear system has more equations than unknowns, and an inconsistent system has no solution.

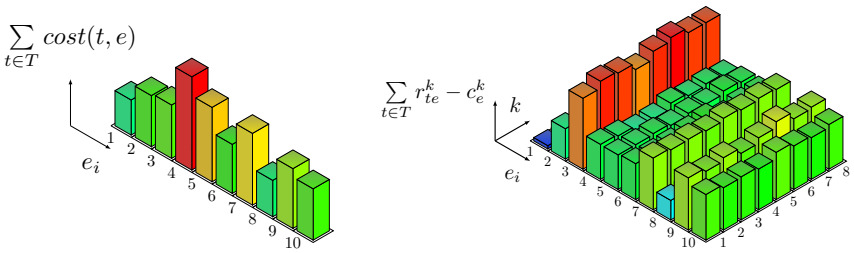
[3] Here the Gauss-Seidel method differs from Jacobi iteration.

with high-valued resources, as a lot of tasks like to claim the resource for the given price. The value x is normalized with respect to the minimal value in x to avoid slow convergence in case of small values (< 1.0), and to avoid early intensification in case of very large values. For each $k \in R$, the penalty is initialized with

$$p_e^k = \frac{x_e}{min_e\, x_e}. \tag{6.13}$$

Evaluation of the initial weights method

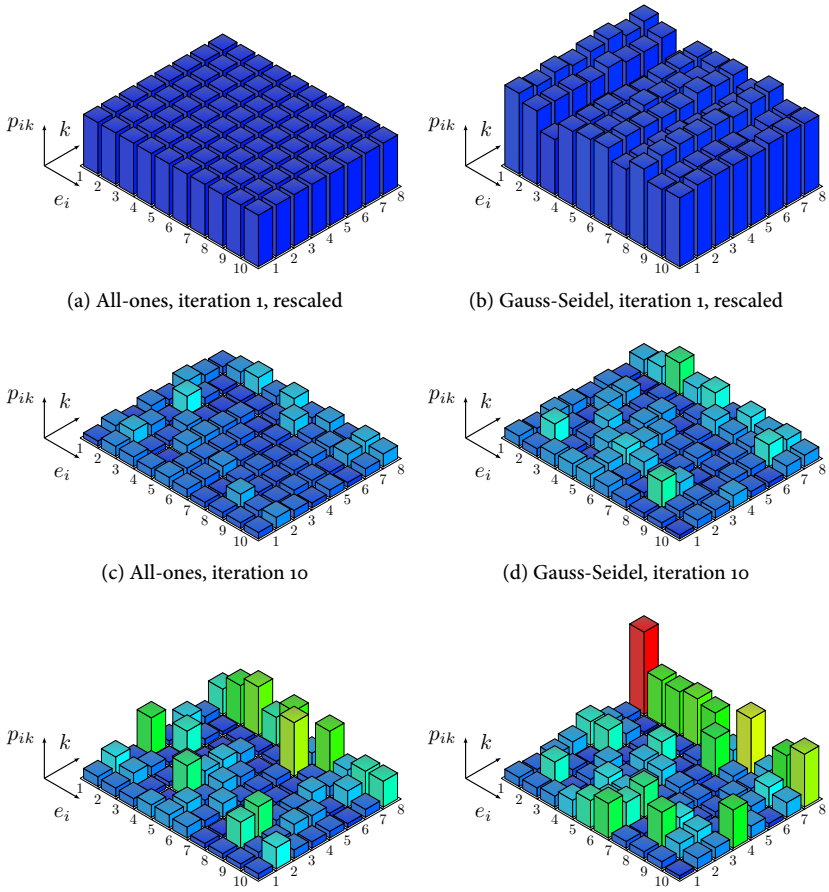
The method of initializing the penalty weight may or may not impact the performance of the optimization process. We provide some insight into the matter with an illustrative problem from the dataset used later in this chapter. Problem e101008 models a platform with 10 elements each having 8 different resource types. One hundred tasks have to be mapped to this platform. Figure 6.6a shows per element e_i the total cost in case all tasks are assigned to that element. This shows that, on average, element 4 is the most costly element to use. Figure 6.6b shows per resource type k the oversubscription if all tasks are assigned to element e_i. This gives insight in the relative resource scarcity in problem e101008. With this problem instance, we compare the Gauss-Seidel based method for setting the initial penalty weights to a uniform initialization with an all-ones matrix. Figure 6.7 shows the relative penalty weights over multiple iterations using either approach. Comparing Figure 6.7a and Figure 6.7b, the Gauss-Seidel method initialized the penalties in relation to Figure 6.6a and Figure 6.6b. This has an effect on the short-term behavior of the local search, where the penalties initialized with the Gauss-Seidel method seem to provide more explicit features, compared to the uniform all-ones approach. This phenomenon can be observed in Figure 6.7c versus Figure 6.7d, and in Figure 6.7e versus Figure 6.7f. In the long run, both initialization methods provide a penalty matrix with useful features; resource type 8 seems to be most critical to solving this problem instance. Resource type 2 and 3 of element e_8 are also penalized above average over many iterations, and with both initialization methods. This is explained by the fact that element e_8 is relatively cheap, and is relatively high on resources. To resolve constraint violations in other parts of the solution, it is then favorable to make use of element e_8. Penalizing these resources ensures that the search is not trapped in local minima due to the inexpensive resources provided by element e_8. With sufficient iterations, the initial difference between both initialization methods has diminished (see Figure 6.7o and Figure 6.7p). Concluding, the all-ones approach starts with a more diverse search, yielding slightly worse solutions earlier in time, while the Gauss-Seidel approach takes a slightly longer intensified search to come up with potentially better results. Due to our focus on the short term and the additional computational effort required for the Gauss-Seidel approach, the all-ones initialization method is used by default.



(a) Potential cost per component.

(b) Potential resource oversubscription.

Figure 6.6: Characteristics of problem instance e101008.



(a) All-ones, iteration 1, rescaled

(b) Gauss-Seidel, iteration 1, rescaled

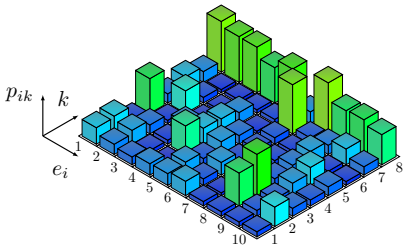
(c) All-ones, iteration 10

(d) Gauss-Seidel, iteration 10

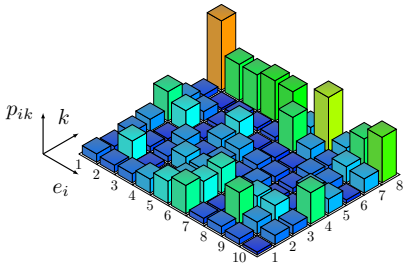
(e) All-ones, iteration 25

(f) Gauss-Seidel, iteration 25

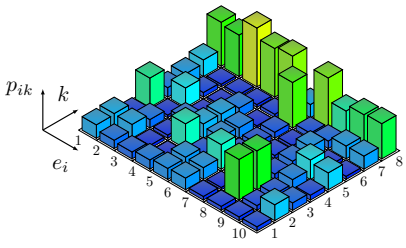
Figure 6.7: Penalty matrix initialized with Gauss-Seidel or all-ones, at various iterations on problem e101008.



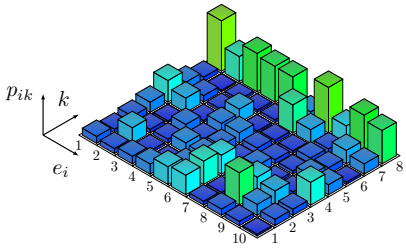
(g) All-ones, iteration 50



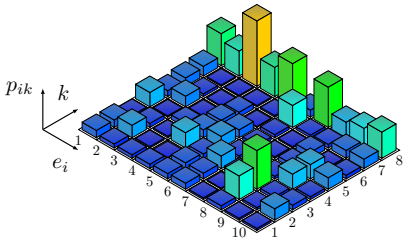
(h) Gauss-Seidel, iteration 50



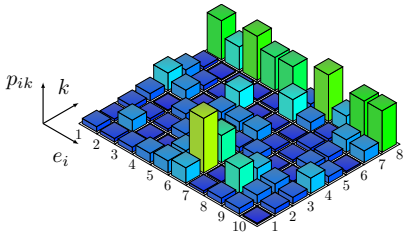
(i) All-ones, iteration 100



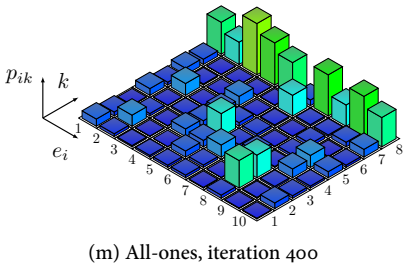
(j) Gauss-Seidel, iteration 100



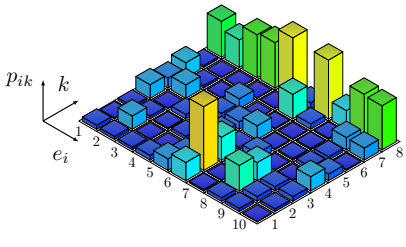
(k) All-ones, iteration 200



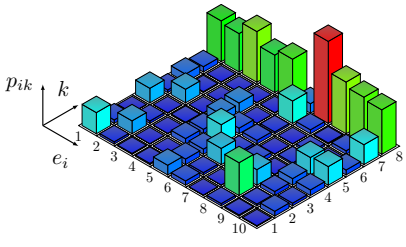
(l) Gauss-Seidel, iteration 200



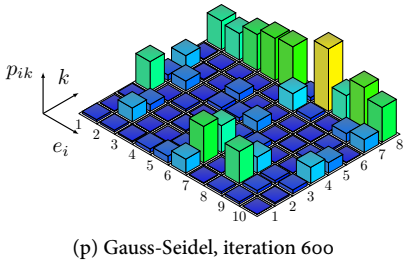
(m) All-ones, iteration 400



(n) Gauss-Seidel, iteration 400



(o) All-ones, iteration 600



(p) Gauss-Seidel, iteration 600

Adaptive weight control

Every local search procedure yields a new locally optimal solution $\bar{\sigma}$. This solution is not necessarily feasible, nor globally optimal. Repeating the local search procedure gives similar results, only influenced by a random factor in the algorithm. The solution $\bar{\sigma}$ has been found using a specific set of penalty weights p, which is based on the resource oversubscription of previous iterations. On account of the penalty weights p, possibly a different set of resources has become oversubscribed. Therefore, we adjust the penalty weights p after each invocation of the local search procedure such that currently oversubscribed resources become more expensive to allocate in the next iteration. Larger penalty weights intensify the search in feasible regions of the search space, while smaller penalties diversify the search towards infeasible regions.

If a feasible solution σ is found, we can continue the search in $\mathcal{N}(\sigma)$ in order to improve σ further. In that case, a small reduction of the penalty weights causes a more fine-grained optimization process, potentially yielding many feasible solutions, at the cost of increased computation time before a (near) optimal solution is found. Alternatively, a more substantial weight reduction results in diversification. The metaheuristic then steers the local search away from a potentially reoccurring difficulty in improving solution σ towards other parts of the search space.

If the last local search iteration yields an infeasible solution $\bar{\sigma} \notin \mathcal{F}$, the penalty weights are increased (Equation 6.14). All resources that are oversubscribed (Equation 6.16), increase their penalties by a factor $(1 + step\_size)$, normalized by the maximum oversubscription.

The procedure for controlling the weights is based on the approach in [122], but is augmented with the dimension in multiple types of resources. Another difference is that we use a variable step size for the weight reduction of the penalties in case of a feasible solution. For certain problems, the search would otherwise spend time on generating suboptimal solutions that are found relatively simple. Instead, Equation 6.17 uses the duality gap defined in Section 6.1.1 to estimate the quality of the incumbent solution. The step size is then controlled by the gap between the incumbent and dual solution, as expressed in Equation 6.17. A larger step size in case of bad quality solutions result in diversification. A smaller step is used when we approach the optimal solution, allowing for a more intensive search. This approach mainly eliminates a slow start for simpler problem instances.

Summarizing, when a local search yields a solution σ, the penalty weights p_e^k are adjusted, per element e and per resource type k as follows:

$$p_e^k = p_e^k \cdot (1 + \Delta q_e^k) \tag{6.14}$$

$$\Delta = \begin{cases} \frac{\text{step\_size}}{max_e\, q_e^k}, & \text{if } max_e\, q_e^k > 0 \\ \text{step\_size}, & \text{otherwise} \end{cases} \tag{6.15}$$

$$q_e^k = \begin{cases} -1, & \text{if } \sigma \in \mathcal{F} \\ max(0, \frac{\sum_{t \in T} r_{te}^k x_{te}}{c_e^k}) & \text{otherwise} \end{cases} \tag{6.16}$$

$$\text{step\_size} = \begin{cases} 1.0 - \frac{Z_D(\sigma)}{Z(\hat{\sigma})}, & \text{if } \hat{\sigma} \neq \varnothing \\ 0.01, & \text{otherwise} \end{cases} \tag{6.17}$$

6.1.4 PATH RELINKING WITH A REFERENCE SET OF SOLUTIONS

Up to this point, we have described a method to find a local optimum for a given solution, taking a weighting function into account to avoid problematic parts of the solution. This procedure can be applied to arbitrary starting points in the search space, keeping track of the incumbent solution. When the local optima of a problem are scattered, intensification of the local search is not sufficient. *Path relinking* is a technique that combines solutions in an attempt to create a 'path' out of a local optimum towards other parts of the search space. This technique is developed by Glover et.al. [40] and proposed for the generalized assignment problem [121], and for the quadratic assignment problem [124]. The path relinking technique is fundamentally different from the rather unstructured *crossover* mechanism often used in evolutionary algorithms [40]. The ability to systematically exploit the neighborhood structures contributes to the improvements of path relinking over alternative metaheuristic algorithms [125].

Reference set

Path relinking requires a reference set $\mathcal{R}$ with a limited number of distinct solutions. Solutions that are (almost) similar to already stored solutions are not so useful. Therefore, we require the solutions stored in the reference set to have a minimal difference d to all other solutions already in the set. This difference can also be interpreted as the distance between solutions. For the MRQAP problem, the distance is naturally defined by the number of different task-to-element mappings in a pair of solutions.

Initially, the reference set is empty ($|\mathcal{R}| = 0$). To populate the reference set, initial solutions are generated as described in Section 6.1.1. These solutions are first improved by applying a local search, such that the resulting solution is guaranteed to be locally optimal. This solution is then added to the reference set, respecting the requirement of diversification (i.e. the minimal distance). Instead of generating random solutions to seed solution set S, the path relinking technique explained in the next section is used whenever the reference set reaches a predefined minimal size ($|\mathcal{R}| >= \zeta$). There is also an upper bound on the size of $\mathcal{R}$, as we only want to use the best n solutions in the path relinking technique. Therefore, the reference set is partially ordered on the penalized cost of the solutions using a min-heap[4].

[4] A min-heap is a binary tree of which the data contained in each node is less than (or equal to) the data in that node's children.

Path relinking

Figure 6.8a illustrates the path relinking technique. Given a populated reference set $\mathcal{R}$, we randomly take one of the best solutions in $\mathcal{R}$, and apply a percentage (in our case 1%) of random shift moves to that solution to ensure diversification of the local search procedure. The resulting solution is referred to with σ_1, shown as white dot in Figure 6.8b. This solution may be, without any restrictions, of a much lower quality compared to the other solutions in $\mathcal{R}$. This randomization mechanism is a tabu search technique that ensures that a different part of the search space is considered during each iteration. Otherwise, choosing the best solution in $\mathcal{R}$ (always) or skipping the randomized moves increases the probability that the search remains in the same local optima in the search space.

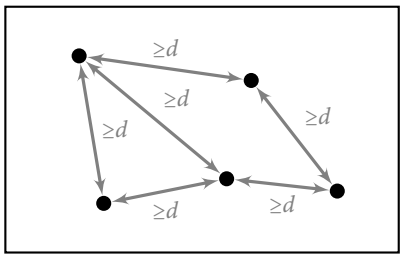
Starting with σ_1, we generate new solutions by combining the best parts of the solutions within $\mathcal{R}$. From every other solution $\sigma_2 \in R$, we take the most beneficial assignment that differs from solution σ_1, i.e. $\langle task, element \rangle \in \sigma_2 \backslash \sigma_1$. Figure 6.8b illustrates this process, where the black and white dots representing solutions are combined, resulting in a new set of solutions made up by the grey dots. Assuming that such an assignment exists for every combination, the size of the resulting set $\mathcal{S}$ of generated solutions is equal to the size of reference set $\mathcal{R}$.

The solutions in $\mathcal{S}$ are likely locally suboptimal due to the changes made by the path relinking procedure. A local search procedure is preformed for every $\sigma \in \mathcal{S}$ to regain a set of locally optimal solutions (Figure 6.8c). These solutions are fed back into the reference set (Figure 6.8d), respecting the previously described requirements on the relation between the solutions in the reference set. Each process solution is taken out of solution set S, which then reduces in size with each local search procedure. Path relinking is done whenever the solution set S becomes empty; that is $S = \varnothing$.

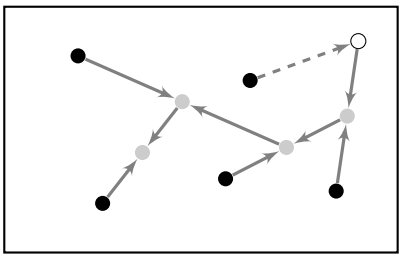
6.1.5 Feedback information

The main issue identified in the approach of Chapter 3 is the lack of information on the reason why an application is rejected when the algorithm fails to find a mapping. Such information would enable run-time changes in the application mode or active application set, and design-time changes in the application resource usage and resources provided by the hardware to avoid the rejection. With the GLS approach, feedback information is available at all times. The penalty weights (matrix) show the relative value of each resource in that specific iteration of the algorithm. Averaging these weights over time compensates for the temporary difficulties in the search and the fluctuations between intensification and diversification of the search. The matrix of average penalty weights provides the required feedback information.

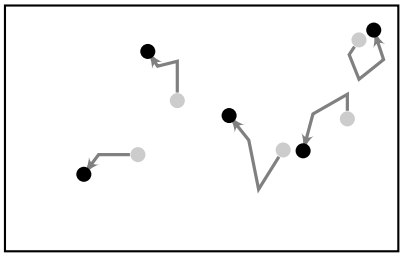
As an example, Figure 6.9 shows the penalty weights of problem instance e101008 summed over all iterations up to and including 1000. Further analysis of the information contained in this matrix needs an additional data aggregation to be useful. A summation over all resource types and all elements provides the following in-



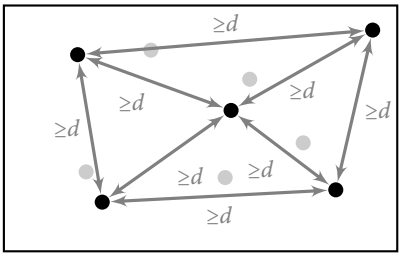
(a) At any point in time, the reference set $\mathcal{R}$ contains local optimal solutions that are minimally d distance apart.



(b) Apply random shifts to 1% of the tasks in a random solution in $\mathcal{R}$ (white dot). Then, generate a set of new solutions $\mathcal{S}$ by combining current locally optimal solutions (path relinking).



(c) Find new locally optimal solutions using shift, swap and chained shift moves.



(d) Renew the reference set of solutions, using the previous $\mathcal{R}$ and the newly found local optima.

Figure 6.8: A visualization of an example search space during one iteration of the tabu search algorithm.

formation. Resource type $k = 8$ and elements e_6 and e_{10} are most problematic to problem instance e101008. Information on the relative scarcity of resource types is considered to be the most valuable, as it requires little further analysis to be used. The system may, for example, change the application QoS levels, release memory, or scale up in voltage and/or frequency. Information about the difficulty in meeting resource constraints at specific hardware elements requires more knowledge about both the architecture and the application. The information presented in Figure 6.6b does not provide obvious reasons for the high penalties put on the use of elements e_6 and e_{10}. However, the feedback information may be a useful addition to the information already known upfront.

With only two vectors containing relative values, the system can continuously report the scarcity of different resource types, and the preference for specific elements in the hardware. These vectors may be used as feedback to upper software layers or controlling entities in the system to adjust the demand for resources on the system.

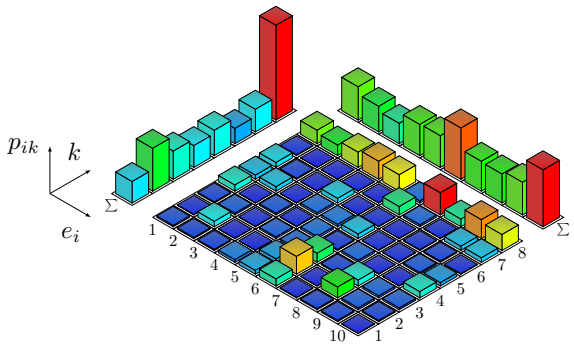


Figure 6.9: Penalty weights p_{ik} of problem instance e101008 summed over time (up to iteration 1000), resource type k and element e_i.

6.1.6 The overall task assignment approach

An overview of the search technique applied to the task assignment subproblem is presented in Figure 6.10. The cycles in the control flow correspond with the iterative behavior of the technique. No termination condition is specified, and no means are available to determine infeasibility or optimality. The search may be terminated when the first solution is found, after a fixed number of iterations, after a fixed time duration, or when a solution of sufficient quality is found using the duality gap as an estimate.

6.2 Communication routing

The approach described so far only considers the task assignment problem. In the MRQARP definition, at least one communication channel[5] per task has to be routed through the interconnect with a predefined resource demand (i.e. bandwidth). The MRQARP formulation allows the interconnect to be an irregular structure, differentiating between links in the interconnect by annotating them with a different weight, reflecting properties such as length (delay) or reliability of a link. More complicated is the assumption that the links in the interconnect have a finite capacity. Virtually, the network changes after each resource allocation or release; parts of the network might even become inaccessible due to congestion. Taking these capacities into account, only a limited set of routing requests can be satisfied. The routing problem is typically described in related work as an optimization problem, where the most profitable subset of the routing requests has to be selected, when the capacity of the network limits the number of requests that can be satisfied. In our case, we have to satisfy all routing requests of the application at hand.

If a set of routing requests cannot be satisfied, the task assignment has to be changed. Similarly, changes in the task assignment require the communication routing to

[5] An application consisting of a single task is an exception.

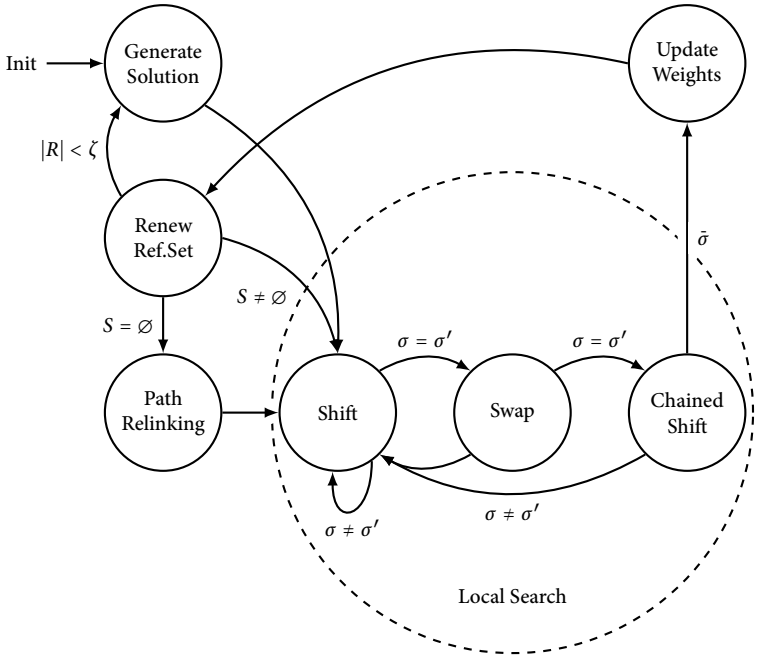


Figure 6.10: Guided local search with shift, swap and chained shift neighborhoods, using path relinking to generate new solutions.

be changed accordingly. In either case, moves in the local search phase reassign tasks to different elements, resulting in invalid routes for the communication channels between tasks. These routes have to be re-established, and the corresponding impact on the required and available resources has to be evaluated.

6.2.1 Integration with the shift, swap and chained shift move

After each shift or swap move in the local search, we have to 'repair' all affected routes. The method described in Section 6.1.2 provides for each possible move in the task assignment a delta d of the change in penalized cost. This d now reflects a cost budget that can be used to reroute these channels. The overall solution will be improved as long as the routing cost remain well within this cost budget. Instead of rerouting all affected communication channels, which is a relative costly operation, an estimation of the routing costs is used by querying a *distance matrix*. This distance matrix contains the costs of routes between all pairs of elements in the platform. A move is thus beneficial for the entire solution, if the combined difference in penalized and estimated cost is an improvement ($d < 0$). The actual rerouting of the channels is deferred after the solution is considered to be locally optimal with respect to the penalized cost of the task assignment and the estimated cost of the routing.

6.2.2 Rerouting Communication Paths

A straightforward approach to solve the routing subproblem of the MRQARPs is to involve many shortest path queries. Finding the shortest paths online (on large graphs) is costly, reducing the scalability of solving MRQARP instances. Alternatively, shortest paths can be computed offline and stored in memory, taking $O(E^2)$ space. This approach may not be feasible for larger graphs due to memory limitations. While the required space can be reduced by exploiting symmetry in graphs [119], they become asymmetric in their properties due to resource allocation. Offline preparation of specialized data structures to improve the performance of shortest path queries is thus nontrivial. Online caching of results of shortest path queries also gives low benefit, because the similarity of queries tends to be quite low. Edges quickly become saturated after a few routing requests, which must be taken into account by future shortest path queries.

Our approach is inspired by [49], where neighborhood operations are defined for local search in the vehicle routing problem [33]. Parts of existing routes are exchanged to form new routes, that potentially improve the total solution. Instead of completely rerouting invalidated routes, we attempt to repair the routes by routing a single source to a set of destinations. The set of destinations is composed by the actual destination and all intermediate nodes of the invalidated route. Figure 6.11 shows the shift move of Section 6.1.2 extended with communication routing. For a shift move involving task t_i, every channel $\langle t_x, t_i \rangle \in C \cup \langle t_i, t_y \rangle \in C$ needs to be rerouted. When shifting task t_i from element e_i to element e_j, each route $\langle e_i, e_y \rangle$ must be adapted by searching the shortest path from element e_j to any element within the set $\{e_y\} \cup \{e_n | e_n \in \langle e_i, \ldots, e_y \rangle\}$. Similarly, the route(s) $\langle e_x, e_i \rangle$ have to be rerouted to any element within the set $\{e_x\} \cup \{e_n | e_n \in \langle e_x, \ldots, e_i \rangle\}$, but taking the reversed direction of the communication links instead. With this approach we benefit from the principle of optimality, where the shortest path problem exhibits *optimal substructure*.

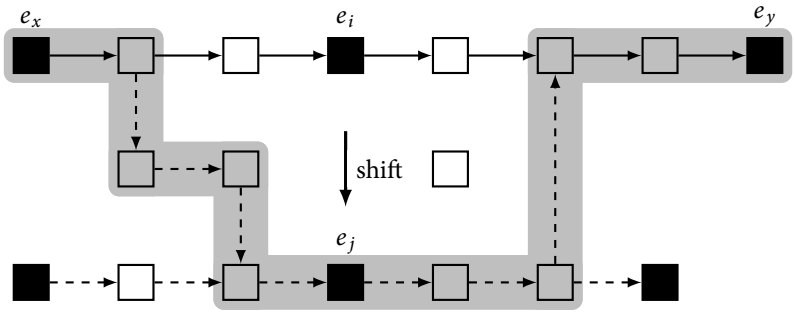


Figure 6.11: When shifting task t_i from element e_i to element e_j, the corresponding routes (solid) from e_x and to e_y have to be rerouted, preferably using parts of the existing routes, resulting in adapted routes (shaded) to the same peers.

Figure 6.12 shows the swap move of Section 6.1.2 extended with communication routing. From the communication routing perspective, a swap move is similar to a shift move, only increasing the number of routes needing repair. Furthermore, the chained shift move can be considered as a specific sequence of swap moves. For both the swap and the chained shift move, the difficulty in exploiting the existing routes is that the tasks involved in the move probably have a different communication topology; i.e. they vary in the amount of input and output channels. In both the swap and the chained shift move, the set of channels that are rerouted compete for the available resources in the interconnect. All resources allocated to the channels involved in the move are virtually released, making them available to any of the channels. Then, the channels are ordered first on the difficulty of routing the channel and second on the amount of resources required. The difficulty of routing a channel is an internal metric, of which the value is increased with each locally optimal solution in which the channel uses oversubscribed resources. Initially, the channels are rerouted while disallowing any resource oversubscription. Any channel that cannot be routed, is routed in a second attempt where resource oversubscription is allowed.

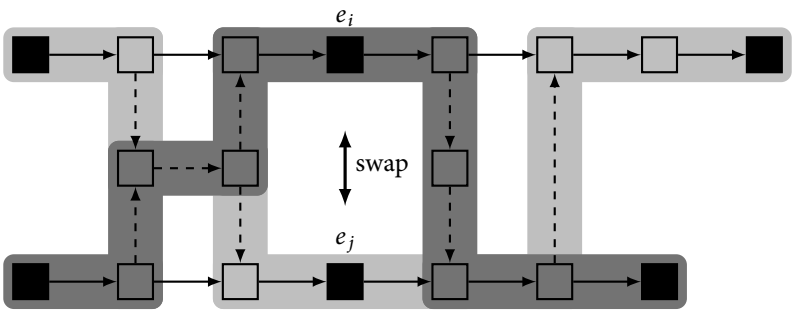


Figure 6.12: When swapping tasks t_i and $t_{i'}$, the routes of two tasks have to be adapted.

6.2.3 TAXATION OF OVERSUBSCRIBED LINKS

Each communication channel that is to be routed through the platform's interconnect aims for the shortest possible path. Selfishly using a route that is perceived to be the shortest path for a communication channel may overall result in congested networks and oversubscription of resources in the network. The degradation in performance of a system due to the selfish behavior of its users, known as the *price of anarchy*, is 2.618 for the unsplittable flow problem [7]. This means that if all communication channels in the application are routed over the shortest path, the observed performance degrades by a factor of 2.618. Other communication problems such as deadlock are not considered in this analysis. On the other hand, adding more communication resources to the platform may also have adverse effects, as the

performance may not always be improving. This phenomenon is known as Braess paradox:

Braess paradox:
For each point of a road network, let there be given the number of cars starting from it, and the destination of the cars. Whether one street is preferable to another depends not only on the quality of the road, but also on the density of the flow. If every driver takes the path that looks most favorable to him, the resultant running times need not be minimal. Furthermore, an extension of the road network may cause a redistribution of the traffic that results in longer individual running times [14].

Given the challenges and potential performance degradation as just described, careful regulation of the network traffic is required. One method is to (virtually) change the properties of the network. The system as a whole may then be improved by increasing the routing cost of some communication links, simultaneously decreasing the routing cost of others. Whenever the routes in a solution $\sigma^{(i)}$ cause oversubscribed resources on links of the interconnect, an incentive to change the routes in the next solution $\sigma^{(i+1)}$ is created by increasing the cost (by adding 'tax') of oversubscribed links. In case of latency-minimization objectives, optimal taxes exist and can be calculated in polynomial time [22]. While the implementation method of [22] is unclear, it shows the usefulness of the mechanism of regulation. This taxation of the network is similar to the penalized cost of oversubscribed resources in the task assignment subproblem. Therefore, we update the cost of both elements and links in the platform in the same procedure, as described in Section 6.1.3. The sole difference is that for the communication links, a fixed step size of 0.01 is used. The step size is fixed due to the lack of any information on the difference between the current solution and the optimal solution. The value of the step size is chosen such that it results in a smooth traversal through the search space without needing too many iterations before other solutions are considered. With each (single channel) routing update, the distance matrix is updated with the current penalized cost. This distance matrix is then used in the local search to estimate the routing cost in the next iteration. A *Wardrop equilibrium* is reached when no communication channel has an incentive to change its assigned route [115]. The solution is then considered to be locally optimal.

6.3 The overall GLS-algorithm

Algorithm 3 provides the pseudo code of our implementation of guided local search for MRQARP. It takes an MRQARP problem instance Z as input. The algorithm consists of two phases. The algorithm aims for a feasible solution in the initialization phase (line 2-11), and attempts to generate improved solutions in the optimization phase (line 12-42). The initialization phase is always executed, while the number of

iterations in the optimization phase depends on the termination condition, which is checked in the beginning of each iteration.

The initialization phase On line 2 the reference set $\mathcal{R}$, the working solution set $\mathcal{S}$ and the incumbent $\hat{\sigma}$ and corresponding upper bound $\hat{Z}$ are initialized. Then, an initial solution is generated using the Lagrangian relaxation technique (line 3). Solution σ' is generated by mainly considering the objective function for Z. It may approximate the optimal solution for Z in terms of cost, but may violate resource constraints. Therefore, a local search procedure within $\mathcal{N}_{shift} \cup \mathcal{N}_{swap}$ is performed that only takes the resource oversubscription penalties into account, and not the cost itself. The shift and swap moves gradually increase the feasibility of solution σ. As a result, the initialization phase may or may not be able to obtain a feasible solution.

The optimization phase Similar to the local search procedure of the initialization phase, a local search procedure is defined in lines 30-41 of the optimization phase. This time, however, the search also traverses $\mathcal{N}_{chain}$ if the working solution σ is of good quality; i.e. when the penalized cost approaches the cost of the incumbent solution $\hat{\sigma}$, accounted by value $\hat{Z}$. When a solution can no longer be improved, it is considered to be locally optimal and the local search is stopped. For MRQARP, the local search takes an approximation of the communication cost into account. In line 41, those communication routes that are invalidated due to changes in the local search procedure are repaired.

At the beginning of each iteration, we ensure that we keep track of the incumbent, associated cost and whether or not the GLS algorithm is to be terminated (lines 13-18). With each locally optimal solution $\bar{\sigma}$, whether feasible or not, the penalty weights p are increased to reflect the difficulties in adhering to the constraints, or decreased when σ is feasible (line 19). Solution σ is added to reference set $\mathcal{R}$ (line 20) if the solution is of sufficient quality and if the solution has enough distinct features to enrich reference set $\mathcal{R}$.

With each iteration, a new local search procedure is started, but with updated penalty weights p, and with a different solution $\sigma' \in S$. If the set of solutions S is empty, it requires repopulation. When reference set $\mathcal{R}$ is sufficiently large, a solution set $\mathcal{S}$ is generated using path relinking (lines 22-24). Otherwise, a random solution is generated (line 25).

Algorithm 3 Guided Local Search

```
1:  procedure GLS(Z)
2:      σ̂, R, S, Ẑ ← nil, ∅, ∅, ∞
3:      σ ← σ′ ← GenerateLRSolution(Z)
4:      p ← InitializeWeights(Z)
5:      repeat                                   ▷ Local search focussing on feasibility
6:          repeat
7:              s ← s′, σ′ ← SearchNshift(Z, σ, p)
8:          until σ′ = σ
9:          σ′ ← SearchNswap(Z, σ, p)
10:     until σ′ = σ
11:     σ ← RepairCommunicationRoutes(Z, σ′, p)          ▷ MRQARP only
12:     loop
13:         if Feasible(Z, σ) ∧ Z(σ) < Ẑ then
14:             Ẑ, σ̂ ← Z(σ), σ
15:         end if
16:         if TerminateCondition( ) then
17:             return σ̂
18:         end if
19:         p ← UpdatePenaltyWeights(Z, σ, p)
20:         R ← AddSolutionToReferenceSet(R, σ)
21:         if S = ∅ then
22:             if |R| >= MinSizeReferenceSet then
23:                 S ← PathRelinking(R)
24:             else
25:                 S ← ∅ ∪ GenerateRandomSolution(Z)
26:             end if
27:         end if
28:         σ′ ← S[0]                    ▷ Take the first solution out of ordered set S
29:         S ← S ∖ σ′
30:         repeat                                   ▷ Local search procedure
31:             repeat
32:                 repeat
33:                     s ← s′, σ′ ← SearchNshift(Z, σ, p)
34:                 until σ′ = σ
35:                 σ′ ← SearchNswap(Z, σ, p)
36:             until σ′ = σ
37:             if PenalizedCost(Z, σ, p) < 1.01 × Ẑ then
38:                 σ′ ← SearchNchain(Z, σ, p)
39:             end if
40:         until σ′ = σ
41:         σ ← RepairCommunicationRoutes(Z, σ′, p)      ▷ MRQARP only
42:     end loop
43: end procedure
```

6.3.1 IMPLEMENTATION

The implementation of GLS requires an input file that specifies the problem to be solved. The solver is incapable of determining the optimal solution, and therefore, will not stop searching. Therefore, one can provide a timeout (in seconds) to limit the allowable search time:

```
$ mrqarp-gls
Usage: mrqarp-gls <input file> [timeout (sec)] [cpu profile] [mem profile]
```


Figure 6.13 shows the output of an execution using problem `c051001` as input, whereas the search time is limited to 5 seconds. A status line is printed whenever the best known solution has improved. The status line gives the elapsed time in seconds, the iteration number and the cost of the incumbent solution. Also, the status line shows the duality gap, and the number of searches in the shift, swap and chained shift neighborhood.

The run shown in Figure 6.13 has reached the timeout of 5 seconds, after which the cost of the best solution found is printed. In this case, the best solution found has cost 1931, which is within 0.4% of the optimal solution. The `mrqarp-gls` solver is implemented in the Go language; due to the footprint of its runtime, the memory usage is measured relative to the baseline memory usage. The maximum memory usage while solving this problem instance is 2056KB.

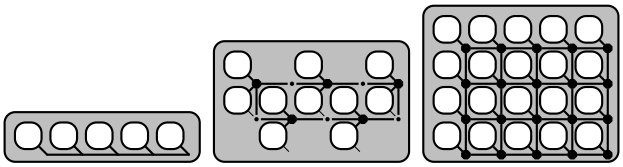
```
$ mrqarp-gls data/mrgap/c051001 5s
Problem: 100 tasks, 5 components, 1 resources, 0 links, 0 channels
Baseline: 32388 kB

Solver:  MRQARP GLS 1.0

 Time (s) | Iteration |  Best  | LB (dual. gap) | Nshift | Nswap | Nchain
------------------------------------------------------------------------------
*    0.01           1     2124   1910.0 (10.1%)        7       4        0
*    0.13           5     1968   1910.0 ( 2.9%)       43      40       15
*    0.18          22     1950   1919.3 ( 1.6%)       75      72       18
*    0.21          23     1937   1920.8 ( 0.8%)       80      77       22
*    0.24          25     1934   1920.8 ( 0.7%)       84      81       26
*    0.36          32     1933   1922.1 ( 0.6%)      102      99       44
*    0.81          71     1931   1923.1 ( 0.4%)      188     185      112
     5.02         455     1931   1923.1 ( 0.4%)      912     909      735
------------------------------------------------------------------------------

Best solution: (cost=1931), Peak usage: 34444 KB, Contribution: 2056 KB
```

Figure 6.13: Output of the solver for an MRGAP instance with 100 tasks, 5 elements, and 1 resource.

Related work on the MRGAP provides a dataset for benchmarking purposes [120, 123, 125], which in turn is based on problem instances from the OR-Library [10, 11]. The dataset is composed of three parts named 'C', 'D' and 'E', where the cost and resource demand for the problems in part 'C' are randomly generated, whereas in parts 'D' and 'E', the cost and resource demand is inversely correlated. Each part contains problems parameterized in their structure, having 100 and 200 tasks, 5, 10, and 20 elements, and 1, 2, 4 and 8 resources per element. This results in 24 problems per part, with 72 problems in total. We extended this dataset to provide MRQARP instances, by generating interconnects for the originally unrelated elements in the dataset, and a communication topology for the tasks. A random number within interval $[0, 2]$ of communication channels is generated per task. Each communication channel receives a bandwidth demand (within interval $[1, 10]$) and uniform costs equal to one. In line with [123] for MRGAP, each link in the generated interconnect provides a bandwidth that is 80% of the summed bandwidth of all channels. This implies that communication links may become oversubscribed when too many channels are routed over a single link. Note that the communication routing might use multiple links per communication channel, increasing the strain on the interconnect. Problems with 5 elements use a bus structure for communication, where the bus is modeled as a hyper-edge [55]. For problems with 10 elements, pairs of elements are attached to a bidirectional ring structure, where the ring is composed of 5 routers. For the larger problems with 20 elements, a 5×4 mesh network is constructed, where the elements are modeled as tiles that are connected to the NoC through means of a router. We denote these datasets with 'CR', 'DR' and 'ER', respectively. See Figure 6.14 for a graphical representation of these platforms.

As we are interested in the short-term performance of the algorithm, we compare the outcome of the GLS algorithm in the time interval (0,10s]. The average solution quality at each sample moment is compared against the commercially available ILP solvers CPLEX 12.5 [60] and Gurobi 5.1 [44]. The ILP solvers are configured to adjust their high-level strategy to prefer good quality solutions over proving optimality. We measure the relative difference between the best found solution and the optimal solution which is known as the *optimality gap*. This should be considered as a measure in terms of relative performance over time, between solvers, and over variations in the problem structure, and not as a measure of absolute quality. In contrast to the ILP solvers, the results of our guided local search algorithm vary per run, caused by a randomization factor in the implementation. Therefore, the results of our algorithm are based on the median of 100 runs for the MRGAP instances, and 30 runs for the MRQARP instances.

As a reference, the best known solution is calculated for every problem in the dataset by running the CPLEX solver exhaustively. For the majority of the problems, a proven optimal solution is found. For the harder problems, optimality could not be proven, and in some cases only a suboptimal solution could be found. This is an indication that the dataset is of sufficient difficulty, as the CPLEX solver took up



(a) bus (b) ring (c) mesh

Figure 6.14: Platforms definitions used in the evaluation.

to 248 hours of computation time[6] for specific problem instances, after which the process was terminated.

6.4.1 Results

The results in this section are obtained on one single-threaded core of a 2.53 GHz Intel P8700 processor with 8 GB of DRAM. For readability purposes, the results are summarized in a set of graphs (Figure 6.15 and Figure 6.16), whereas the detailed measurements are provided in Appendix C.

Performance

Figure 6.15 shows the convergence characteristics of problem instances 're051002' and 'rd101004' for the first two seconds of execution. Problem instance 're051002' consists of 5 elements (connected via a bus), 100 tasks and 2 resource types, whereas problem 'rd101004' contains 10 elements (in a ring topology), 100 tasks and 4 resource types. For both problems, the results for the GLS method are deterministic in the initial part of the run. The Lagrangian relaxation method is used as a seed to obtain feasible solutions first, which contains no randomization factors. When the GLS goes into the optimization mode in order to find better solutions, randomization is used for diversification purposes; this is the cause of some variability between runs. The observed variation is captured with the 5^{th} and 95^{th} percentile, denoted with X_5 and X_{95} respectively. The area between X_5 and X_{95} is shaded.

The graphs in Figure 6.16 show the aggregated results of our GLS implementation on the 12 problems in dataset 'C', 'D', 'E', 'CR', 'DR', and 'ER'. On average, the GLS approach yields feasible solutions within a hundred milliseconds, often within 10% of the optimal solution. The best five percent (X_5) of the solutions from the GLS are, on average, always better than the results of the ILP solvers, while the median of the results is competitive.

For the datasets with the bus interconnect ($|E| = 5$), the problem remains a complex integer packing problem, which is in favor of the ILP solvers. All solvers have similar convergence characteristics for the ring structure ($|E| = 10$), with GLS being slightly

[6] On an Intel Xeon CPU E5645 @ 2.40GHz.

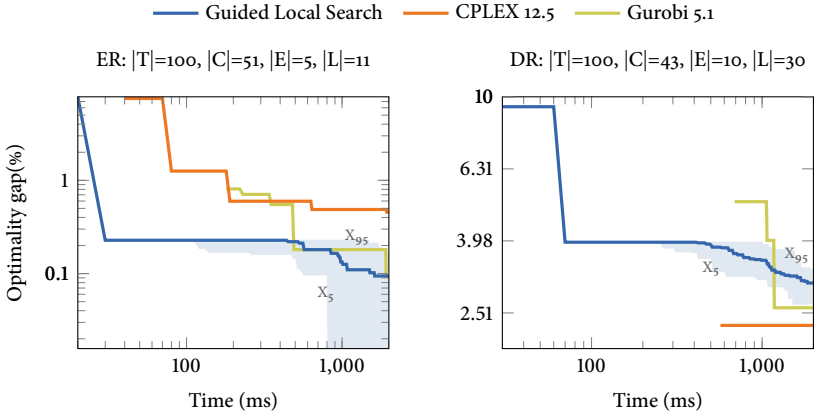


Figure 6.15: Median convergence characteristics of GLS, CPLEX and Gurobi on the er051002 (left) and the dr101004 (right) problem instances.

faster to the first feasible solution. The problems with the mesh networks ($|E| = 20$) put more strain on the ILP solvers. GLS clearly outperforms both ILP solvers in the first seconds of the optimization process. The GLS approach is, on average, always first in finding an initial solution, except for problem instance er051008. The detailed results listed in Appendix C show that in 78% of the cases, the GLS approach outperforms the ILP solvers in terms of solution quality at the moments when an initial feasible solution of any of the ILP solvers becomes available.

Memory usage

By definition, the run-time mapping problem is to be solved at run-time. The allowed footprint of any approach solving the problem should be reasonable for the targeted systems. The memory usage of the solvers used in the benchmarking procedure is measured. These measurements give the upper bound on the memory that should be available at a certain point in time. The peak memory usage is given in Table 6.1 for the MRGAP instances and in Table 6.2 for the MRQARP instances. The required memory scales with the size of the platform; both in the dimension of the number of elements, as well as with the complexity of the interconnect. In any case, the memory usage of the ILP solvers is one or two orders of magnitude larger compared to the GLS approach. The amount of memory required might still be trimmed down, but it is expected that it will negatively influence the response-time of the algorithm. This is especially valid for the ILP solvers, which need to store a set of nodes of a branching tree, where the number of nodes increase with the size of the problem and increase with the number of iterations of the search process. While the memory required by the GLS approach also scales with the size of the input problem, the memory requirements remain constant over time due to the iterative nature of the algorithm.

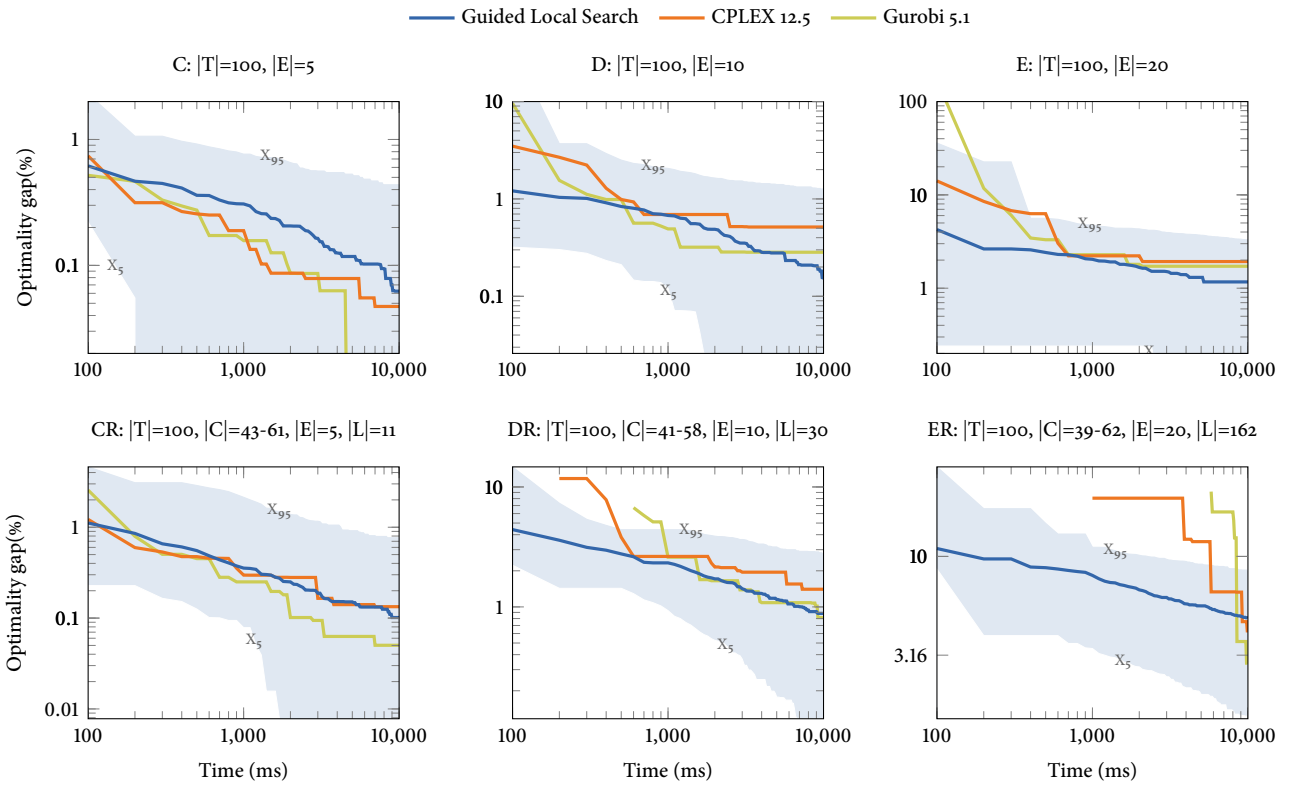


Figure 6.16: Median convergence characteristics of GLS, CPLEX and Gurobi on MRGAP and MRQARP problem instances.

Table 6.1: Peak memory usage while solving MRGAP instances (MB).

Solver	C_{05}	D_{05}	E_{05}	C_{10}	D_{10}	E_{10}	C_{20}	D_{20}	E_{20}
GLS	1.4	1.4	1.6	1.9	1.8	1.9	2.7	2.8	2.7
CPLEX 12.5	11.6	15.6	13.3	12.8	25.8	24.9	14.2	43.5	30.7
Gurobi 5.1	10.9	19.3	12.5	15.4	20.4	20.0	30.8	36.9	37.4

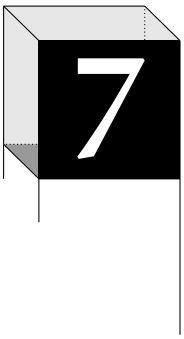
Table 6.2: Peak memory usage while solving MRQARP instances (MB).

	bus			ring			mesh		
Solver	CR_{05}	DR_{05}	ER_{05}	CR_{10}	DR_{10}	ER_{10}	CR_{20}	DR_{20}	ER_{20}
GLS	1.8	1.9	1.9	2.9	3.0	2.9	6.1	6.2	6.1
CPLEX 12.5	15.8	29.7	20.2	32.9	34.0	34.1	266.2	279.7	268.0
Gurobi 5.1	34.7	68.2	36.8	131.5	161.1	136.4	1178.4	1471.8	1206.9

6.5 Conclusions

The proposed GLS algorithm for the MRQARP is robust with respect to all 72 problem instances in the dataset, and provides an overall balance between good initial solutions and a stable convergence to the optimum. On the short term, GLS outperforms state-of-the-art ILP solvers when the scale of the platform and interconnect increases. For the problem instances in the dataset, GLS is able to obtain solutions within 10% of the optimum with only a few megabytes of memory and within hundreds of milliseconds. An additional reduction in the run-time overhead may be obtained with a hybrid mapping strategy that makes use of configurations precomputed at design-time. This hypothesis is based on the use of a reference set for path relinking, which may resemble a set of pre-calculated mappings at design-time. Validation of the hybrid mapping approach is considered to be future work.

The proposed GLS procedure is an anytime algorithm that is able to provide over time a sequence of increasingly better solutions. This enables a system incorporating the GLS algorithm to adapt to the operational requirements, and allows for in-system optimization of long-running configurations. Embedded in an enclosing framework, the GLS algorithm is able to provide feedback information on the relative scarcity of a resource.



7 Conclusions and recommendations

The thesis was set out to explore improvements on the flexibility of large scale heterogeneous and embedded systems by postponing decisions on the allocation of resources to applications from design-time to run-time. The obtained flexibility can be exploited to increase the degree of fault tolerance, quality of service, energy efficiency and to support a higher variability in application structure and use-cases.

The thesis that this dissertation has supported is that *run-time mapping of streaming applications onto large-scale heterogeneous embedded systems is feasible and gives improved flexibility compared to design-time mapping.*

The feasibility of run-time mapping was demonstrated by presenting two methods that solve the resource allocation problem at run-time. The first method is a deterministic algorithm with domain specific heuristics. This technique is implemented in a prototype run-time system, demonstrated with the hardware developed in the CRISP project, and used in subsequent demonstrators for the STARS project. The improved flexibility over design-time mapping was shown with a prototype run-time mapping system, which proved to be able to generate various configurations for GNSS, and three beamforming applications. Simulated hardware faults that would otherwise cause a design-time generated resource allocation to fail, are circumvented by the run-time mapping system. Increasing the degree of flexibility even further, the run-time system is leveraged to perform a health check at startup to verify the correct operation of each individual resource, being a complete processor or a switch-path inside a network router.

The time required for the startup of run-time mapping enabled application varies, from a user perspective, between hundreds of milliseconds up to a few seconds. On one hand, requirements on the startup time could not easily be formulated by industrial partners because it has never been considered to be an important parameter due to the long running times of applications. On the other hand, the startup time seems long compared to nowadays desktop computing. The long startup times reported can be, in some cases, attributed to the large scale of the problems, and in other cases to the difficulty of the (benchmark) problems.

The formulation of the run-time mapping problem and its relation to existing knowledge in the domain of discrete mathematics and operations research shows that the presented combination of the subproblems of task mapping and communication routing, both with limited capacities, is not well studied. The main reasons for the limited availability of fundamental work on this problem is both the theoretical complexity as well as the many simplifications to the problem researcher typically make due to domain specific assumptions. Indicative to this complexity is that it is challenging to concisely and clearly visualize the state of the system together with all properties relevant to the resource allocation problem.

This thesis used empirical findings to show that deterministic mapping algorithms have their weaknesses when it comes to robustness. One of the most noteworthy issues to consider is that a global view on the problem being solved is required to break symmetries in the problem and to avoid being stuck in infeasible areas of the search space. Mismatches between applications and platforms may otherwise give unexpected rejection of resource requests. Moreover, in case of a rejection, the system is not able to provide useful feedback on the reasons of the rejection. Not only the capabilities of the system have been reduced in that case, but also the system is impeded to take action to regain the lost capabilities.

Therefore, starting off with problem solving techniques for related problems presented in the domain of operations research, a second method for resource allocation, a randomized approach, is developed to strengthen the view on the global aspects of the problem. In contrast to the deterministic technique presented first, the randomized approach, known as guided local search, avoids topological orderings of either application or platform graphs. The quality of the results are competitive with industrial ILP solvers, which require a large amount of memory to be performant. It is interesting to note that this approach is able to provide some feedback on the difficulty in fulfilling parts of the resource request; the relative scarcity of specific types of resources is provided together with an indication of the most problematic parts of the application graph.

The characteristics of the guided local search approach make it a suitable candidate to construct a hierarchical resource management system to handle systems of large scale. Only a single implementation of the resource allocation algorithms is required. At each layer, the algorithms work on a different level of granularity. Nodes at lower layers provide useful feedback to the upper layers on the current state of the system, and even on the current state of the request being processed.

7.1 Recommendations for future research

In this section, four different areas relating to this thesis are recommended for future research. In short, this entails field testing of the GLS approach, synthesis of operational modes of a system to cost functions, overhead reduction by hybrid mapping and increased flexibility using decompositional performance analysis.

7.1.1 FIELD TESTING OF GUIDED LOCAL SEARCH

One of the limitations on the evaluation of the GLS approach is the use of a synthetic benchmark set. The benchmark used is known in literature, and is considered to be relatively hard, even outside the context of embedded systems. On one hand, it is expected that real world performance is not slower due to the overdimensioning of the available resources in the system, making the problems less hard. On the other hand, the problems contained in the dataset likely expose less symmetry in platform and application structure compared to real world scenarios. Symmetry in the problem may cause repetitive evaluation of almost identical resource allocations, resulting in more computational overhead. Field testing of the GLS approach improves the insight in the overhead to be expected in real world scenarios.

7.1.2 COST MODELS

Another aspect for further evaluation is the definition of the cost functions used. Whereas our deterministic and domain specific approach evaluated various definitions, the randomized approach of GLS does not. The advantage of the latter is its wide applicability and support for various optimization objectives. The synthetic benchmark used in the evaluation of the GLS already contained the required cost values. An interesting research direction would be to examine sensible cost models for real systems, and to see if those models can be synthesized in the form proposed in this thesis. Additionally, various strategies or operational modes of the system can be defined, resulting in multiple cost functions. Only one cost function is used at a time, which is swapped for another cost function if the strategy or mode of the system changes. This would lay the basis for an adaptable or self organizing system.

7.1.3 HYBRID MAPPING

A hybrid mapping strategy is suggested in order to reduce the overhead of computing a resource allocation from scratch. While this concept has not been evaluated as such, especially not in the context of a real system, a similar mechanism has proven to be useful. The path relinking technique proposed in this thesis uses a reference set of good solutions found during the search process. This reference set is used as a seed for the local search procedure to explore different areas of the search space. Likewise, a design-time generated set of solutions can be used as a seed. A solution in that set may be used if the solution is feasible; or, the solution may be altered to optimize the resource allocation with respect to the actual situation and to work around unforeseen scenarios, such as hardware faults. In case of hybrid mapping, the GLS approach can thus be used as a fallback mechanism or as an in-system optimization engine.

7.1.4 DECOMPOSITIONAL PERFORMANCE SYNTHESIS

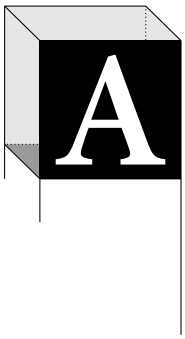
Design-time performance analysis is commonly used to determine the appropriate configuration of an application at run-time in order to meet its timing constraints,

often expressed as minimum throughput or a maximum end-to-end latency. In certain research communities, dataflow analysis techniques are commonly applied to derive scheduler settings (time budgets) for tasks and buffer sizes for communication channels between tasks that match the required performance. As a result, the attributes of the graphs modeling these applications are optimized for a Pareto point in a multi-objective function; that is, an optimized combination of performance versus the required resources.

The increased scale of the next-generation hardware platforms and the varying amount of resources available to each application makes such design-time performance analysis techniques less suitable for flexible embedded systems. Additional flexibility is gained if performance is inter-exchangeable. For example, the first task in a 2-stage pipeline may be allowed to reduce in performance if that loss is compensated by the second task. Whereas compositionality is the foundation of component-based *system composition* in software engineering, decompositionality is the foundation of component-based *system specification* [37]. So, we propose decompositional performance constraints to serve as a specification while the application graph is being constructed and resources are being allocation accordingly. Future work thus includes the formalization of the decomposition rules of performance analysis in order to investigate any integration with the resource allocation problem.

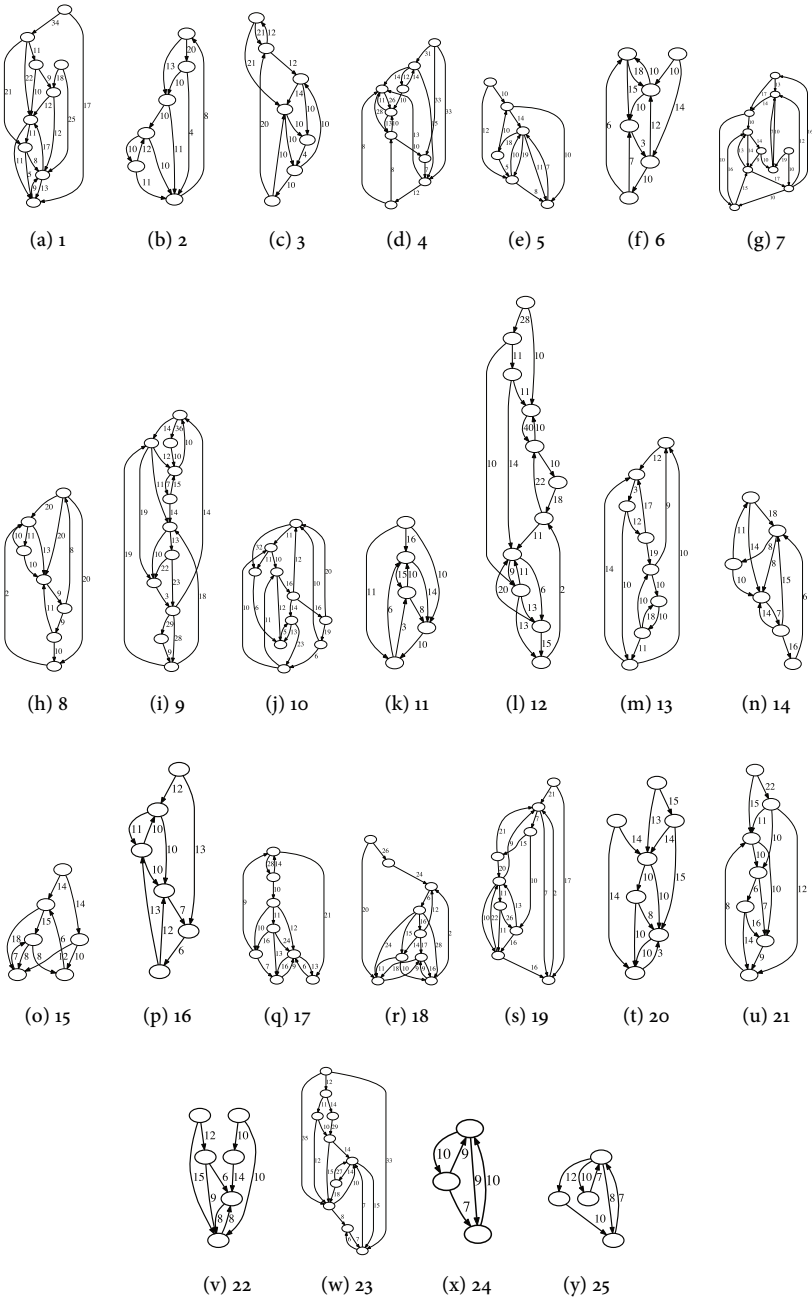
7.2 Concluding remarks

The introduction of full fledged run-time mapping systems has long been delayed due to the inherent complexity of the problems to be solved. While similar mapping problems are already solved at design-time, different analysis and problem solving techniques are required at run-time. The guided local search technique presented in this thesis provides a balance between robustness and acceptable overhead, while being sufficiently generic to be integrated as a key component in large scale embedded systems.



Synthetic benchmark

An in-house developed application generator, similar to TGFF [28], is used to generate synthetic applications. In this tool, the structure of an application can be specified with a number of input, internal, and output tasks. Also the maximum in-degree and out-degree of tasks gives direction to the generated communication structure. The structure of the generated applications can be found in Figure A.1.



(a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6 (g) 7

(h) 8 (i) 9 (j) 10 (k) 11 (l) 12 (m) 13 (n) 14

(o) 15 (p) 16 (q) 17 (r) 18 (s) 19 (t) 20 (u) 21

(v) 22 (w) 23 (x) 24 (y) 25

Figure A.1: Synthetic application graphs generated for the synthetic benchmark.

B

Mapping applications on the CRISP platform

B.1 Application graphs

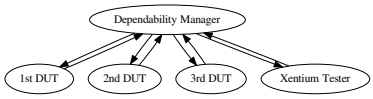


Figure B.1: Dependability tester.

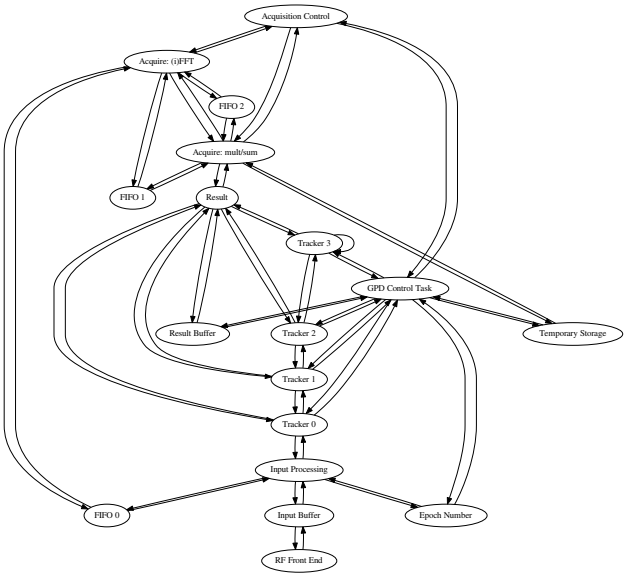


Figure B.2: GNSS receiver.

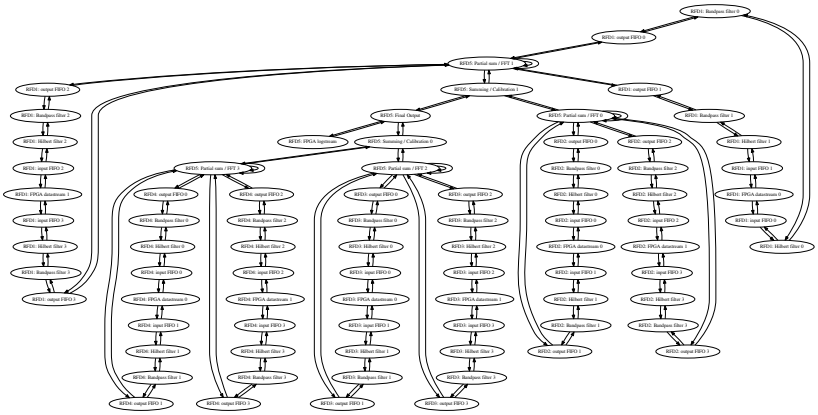


Figure B.3: 16-channel digital beamformer.

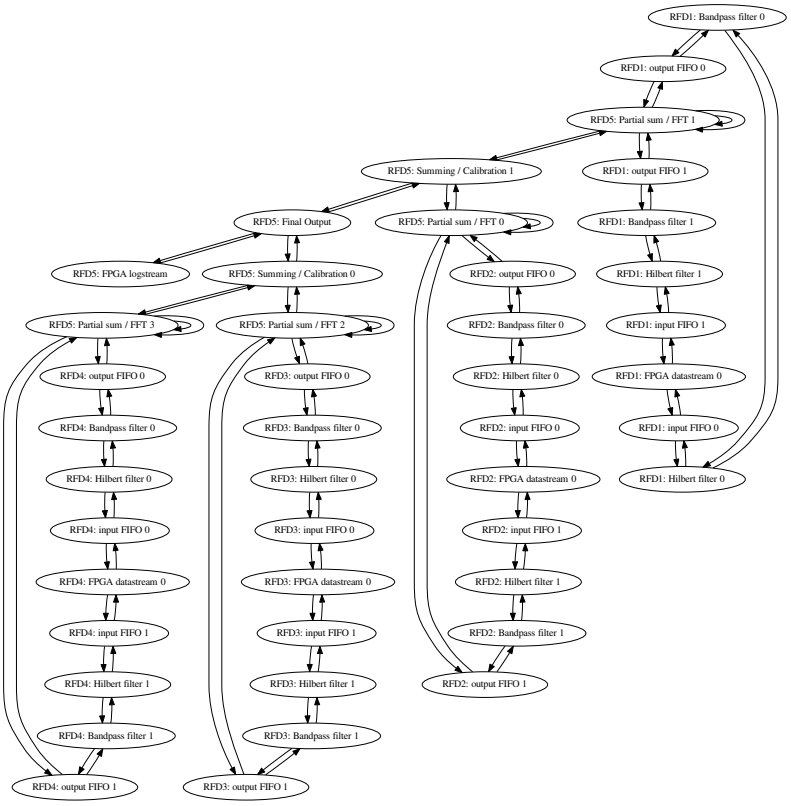
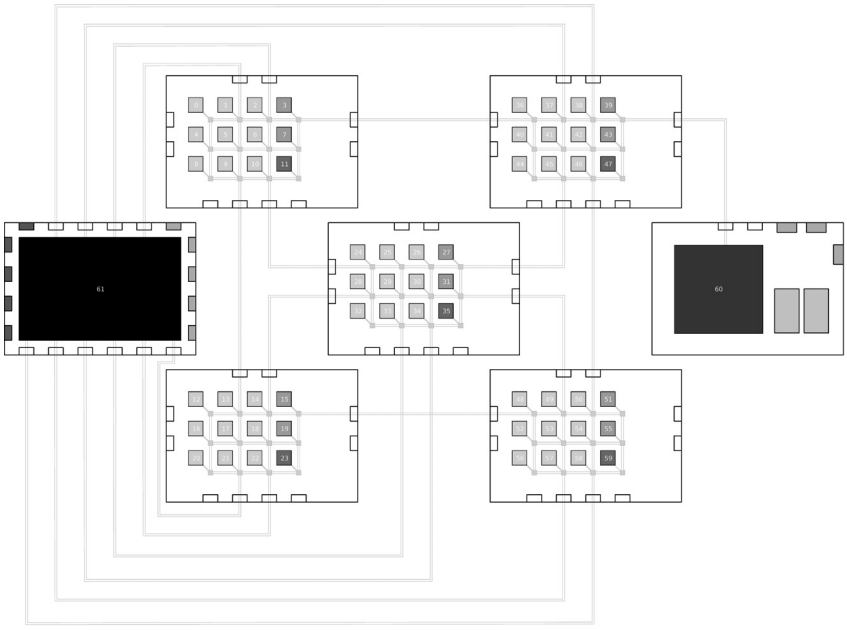
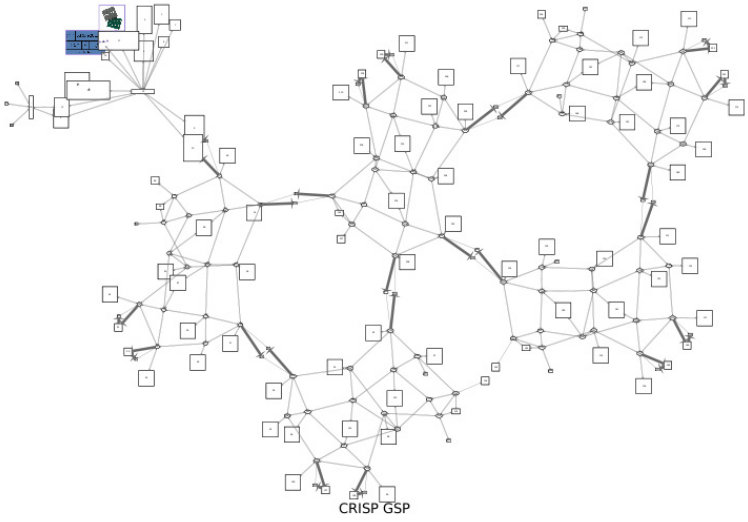


Figure B.4: 8-channel digital beamformer.

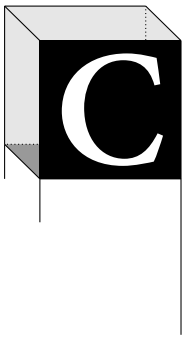


(a) Handcrafted platform visualization.



(b) Automated platform visualization.

Figure B.5: Visualization of the CRISP platform model.



Benchmark results of the GLS algorithm

The tables in this appendix provide the numerical results of the benchmark described in Section 6.4, of which the results are visualized in Figure 6.16. For readability purposes, each table provides a summary in the bottom row. The summary gives per time window (column) the number of test cases for which it was able to obtain at least one solution. Next, it provides the optimality gap, averaged over the amount of test cases contained in the specified dataset. The lower bound is established by running Gurobi exhaustively (for up to 7 days on a single problem) in an attempt to provide the optimal solution to the problem. This attempt failed at more complex problems; the lower bound of these problems have been marked as such.

Table C.1: Solution quality over time with GLS, CPLEX and Gurobi for dataset C-100-*

Problem i	j	k	LB	■ = Guided Local Search ■ = CPLEX 12.5 ■ = Gurobi 5.1 < 100*ms*	< 250*ms*	< 500*ms*	< 1000*ms*	< 2000*ms*
100	05	1	1931	1942 (0.56%)	1938 (0.36%)	1935 (0.20%)	1933 (0.10%)	1931 (0.00%)
						1931 (0.00%)	1931 (0.00%)	1931 (0.00%)
				1931 (0.00%)	1931 (0.00%)	1931 (0.00%)	1931 (0.00%)	1931 (0.00%)
100	05	2	1933	1945 (0.62%)	1938 (0.25%)	1933 (0.00%)	1933 (0.00%)	1933 (0.00%)
				1945 (0.62%)	1933 (0.00%)	1933 (0.00%)	1933 (0.00%)	1933 (0.00%)
				1933 (0.00%)	1933 (0.00%)	1933 (0.00%)	1933 (0.00%)	1933 (0.00%)
100	05	4	1943	1954 (0.56%)	1952 (0.46%)	1949 (0.30%)	1949 (0.30%)	1947 (0.20%)
				1949 (0.30%)	1949 (0.30%)	1949 (0.30%)	1944 (0.05%)	1944 (0.05%)
				1949 (0.30%)	1949 (0.30%)	1944 (0.05%)	1943 (0.00%)	1943 (0.00%)
100	05	8	1950	1957 (0.35%)	1957 (0.35%)	1957 (0.35%)	1957 (0.35%)	1955 (0.25%)
				1977 (1.38%)	1955 (0.25%)	1955 (0.25%)	1952 (0.10%)	1952 (0.10%)
				1964 (0.71%)	1957 (0.35%)	1957 (0.35%)	1952 (0.10%)	1952 (0.10%)
100	10	1	1402	1414 (0.85%)	1410 (0.57%)	1407 (0.35%)	1405 (0.21%)	1403 (0.07%)
				1403 (0.07%)	1402 (0.00%)	1402 (0.00%)	1402 (0.00%)	1402 (0.00%)
				1410 (0.57%)	1406 (0.28%)	1406 (0.28%)	1402 (0.00%)	1402 (0.00%)
100	10	2	1409	1414 (0.35%)	1414 (0.35%)	1413 (0.28%)	1411 (0.14%)	1411 (0.14%)
				1462 (3.76%)	1418 (0.63%)	1418 (0.63%)	1409 (0.00%)	1409 (0.00%)
				1544 (9.58%)	1419 (0.70%)	1411 (0.14%)	1409 (0.00%)	1409 (0.00%)
100	10	4	1419	1430 (0.77%)	1430 (0.77%)	1428 (0.63%)	1425 (0.42%)	1422 (0.21%)
				1457 (2.67%)	1422 (0.21%)	1422 (0.21%)	1419 (0.00%)	1419 (0.00%)
				1571 (10.71%)	1441 (1.55%)	1433 (0.98%)	1419 (0.00%)	1419 (0.00%)

100	10	8	1435	1459 (1.67%)	1453 (1.25%)	1447 (0.83%)	1442 (0.48%)	1439 (0.27%)
				1485 (3.48%)	1485 (3.48%)	1485 (3.48%)	1440 (0.34%)	1440 (0.34%)
				1527 (6.41%)	1451 (1.11%)	1441 (0.41%)	1438 (0.20%)	1436 (0.06%)
100	20	1	1243	1263 (1.60%)	1261 (1.44%)	1255 (0.96%)	1250 (0.56%)	1248 (0.40%)
					1260 (1.36%)	1243 (0.00%)	1243 (0.00%)	1243 (0.00%)
				2033 (63.55%)	1263 (1.60%)	1248 (0.40%)	1243 (0.00%)	1243 (0.00%)
100	20	2	1250	1253 (0.24%)	1253 (0.24%)	1253 (0.24%)	1253 (0.24%)	1253 (0.24%)
					1429 (14.32%)	1343 (7.44%)	1261 (0.88%)	1250 (0.00%)
					1358 (8.64%)	1266 (1.28%)	1250 (0.00%)	1250 (0.00%)
100	20	4	1254	1365 (8.85%)	1261 (0.55%)	1261 (0.55%)	1260 (0.47%)	1259 (0.39%)
				1431 (14.11%)	1431 (14.11%)	1258 (0.31%)	1258 (0.31%)	1254 (0.00%)
				3026 (141.30%)	1330 (6.06%)	1272 (1.43%)	1272 (1.43%)	1255 (0.07%)
100	20	8	1267	1324 (4.49%)	1290 (1.81%)	1281 (1.10%)	1274 (0.55%)	1272 (0.39%)
				3183 (151.22%)	1304 (2.92%)	1304 (2.92%)	1270 (0.23%)	1269 (0.15%)
				3040 (139.93%)	1455 (14.83%)	1455 (14.83%)	1314 (3.70%)	1267 (0.00%)
Guided Local Search				12/12 (1.74%)	12/12 (0.70%)	12/12 (0.48%)	12/12 (0.31%)	12/12 (0.21%)
CPLEX 12.5				9/12 (19.73%)	11/12 (3.41%)	12/12 (1.29%)	12/12 (0.15%)	12/12 (0.05%)
Gurobi 5.1				11/12 (33.91%)	12/12 (2.95%)	12/12 (1.67%)	12/12 (0.44%)	12/12 (0.01%)

Table C.2: Solution quality over time with GLS, CPLEX and Gurobi for dataset D-100-*

Problem i	j	k	LB	■ = Guided Local Search ■ = CPLEX 12.5 ■ = Gurobi 5.1 < 100*ms*	< 250*ms*	< 500*ms*	< 1000*ms*	< 2000*ms*
100	05	1	6353	6403 (0.78%)	6392 (0.61%)	6384 (0.48%)	6376 (0.36%)	6372 (0.29%)
				6373 (0.31%)	6373 (0.31%)	6365 (0.18%)	6365 (0.18%)	6363 (0.15%)
				6374 (0.33%)	6374 (0.33%)	6368 (0.23%)	6368 (0.23%)	6361 (0.12%)
100	05	2	6355	6402 (0.73%)	6396 (0.64%)	6390 (0.55%)	6383 (0.44%)	6376 (0.33%)
				6373 (0.28%)	6373 (0.28%)	6368 (0.20%)	6368 (0.20%)	6358 (0.04%)
				6388 (0.51%)	6377 (0.34%)	6365 (0.15%)	6365 (0.15%)	6365 (0.15%)
100	05	4	6369	6432 (0.98%)	6415 (0.72%)	6408 (0.61%)	6403 (0.53%)	6398 (0.45%)
				6385 (0.25%)	6385 (0.25%)	6385 (0.25%)	6385 (0.25%)	6385 (0.25%)
				6419 (0.78%)	6403 (0.53%)	6393 (0.37%)	6380 (0.17%)	6380 (0.17%)
100	05	8	6406	6474 (1.06%)	6474 (1.06%)	6468 (0.96%)	6457 (0.79%)	6452 (0.71%)
				6531 (1.95%)	6483 (1.20%)	6452 (0.71%)	6452 (0.71%)	6446 (0.62%)
				6499 (1.45%)	6470 (0.99%)	6470 (0.99%)	6463 (0.88%)	6461 (0.85%)
100	10	1	6347	6598 (3.95%)	6546 (3.13%)	6501 (2.42%)	6467 (1.89%)	6443 (1.51%)
				6470 (1.93%)	6382 (0.55%)	6381 (0.53%)	6381 (0.53%)	6380 (0.51%)
				6455 (1.70%)	6365 (0.28%)	6365 (0.28%)	6365 (0.28%)	6365 (0.28%)
100	10	2	6358	6424 (1.03%)	6424 (1.03%)	6420 (0.97%)	6410 (0.81%)	6406 (0.75%)
				6501 (2.24%)	6501 (2.24%)	6402 (0.69%)	6402 (0.69%)	6402 (0.69%)
				6580 (3.49%)	6458 (1.57%)	6444 (1.35%)	6432 (1.16%)	6401 (0.67%)
100	10	4	6374	6484 (1.72%)	6484 (1.72%)	6479 (1.64%)	6470 (1.50%)	6459 (1.33%)
				6516 (2.22%)	6516 (2.22%)	6469 (1.49%)	6469 (1.49%)	6469 (1.49%)
				6570 (3.07%)	6497 (1.92%)	6497 (1.92%)	6497 (1.92%)	6469 (1.49%)

100	10	8	⌊6401	7035 (9.90%)	6545 (2.24%)	6540 (2.17%)	6510 (1.70%)	6500 (1.54%)
				6674 (4.26%)	6674 (4.26%)	6674 (4.26%)	6498 (1.51%)	6498 (1.51%)
				7429 (16.05%)	6568 (2.60%)	6568 (2.60%)	6562 (2.51%)	6562 (2.51%)
100	20	1	⌊6167	6428 (4.23%)	6428 (4.23%)	6428 (4.23%)	6379 (3.43%)	6340 (2.80%)
				6355 (3.04%)	6355 (3.04%)	6355 (3.04%)	6303 (2.20%)	6303 (2.20%)
				6337 (2.75%)	6337 (2.75%)	6337 (2.75%)	6272 (1.70%)	6272 (1.70%)
100	20	2	⌊6172	6368 (3.17%)	6358 (3.01%)	6332 (2.59%)	6318 (2.36%)	6301 (2.09%)
				6359 (3.02%)	6359 (3.02%)	6359 (3.02%)	6335 (2.64%)	6335 (2.64%)
				7304 (18.34%)	6376 (3.30%)	6376 (3.30%)	6313 (2.28%)	6313 (2.28%)
100	20	4	⌊6188	6350 (2.61%)	6350 (2.61%)	6350 (2.61%)	6341 (2.47%)	6330 (2.29%)
				6576 (6.27%)	6576 (6.27%)	6576 (6.27%)	6576 (6.27%)	6421 (3.76%)
				7454 (20.45%)	6501 (5.05%)	6501 (5.05%)	6436 (4.00%)	6395 (3.34%)
100	20	8	⌊6223	6987 (12.27%)	6466 (3.90%)	6466 (3.90%)	6466 (3.90%)	6454 (3.71%)
					6645 (6.78%)	6645 (6.78%)	6645 (6.78%)	6568 (5.54%)
				7527 (20.95%)	7527 (20.95%)	6711 (7.84%)	6711 (7.84%)	6545 (5.17%)
Guided Local Search				12/12 (3.53%)	12/12 (2.07%)	12/12 (1.92%)	12/12 (1.68%)	12/12 (1.48%)
CPLEX 12.5				11/12 (2.34%)	12/12 (2.53%)	12/12 (2.28%)	12/12 (1.95%)	12/12 (1.61%)
Gurobi 5.1				12/12 (7.48%)	12/12 (3.38%)	12/12 (2.23%)	12/12 (1.92%)	12/12 (1.56%)

⌊ The (lower) bound is not tight.

Table C.3: Solution quality over time with GLS, CPLEX and Gurobi for dataset E-100-*

Problem i	j	k	LB	■ = Guided Local Search ■ = CPLEX 12.5 ■ = Gurobi 5.1 < 100*ms*	< 250*ms*	< 500*ms*	< 1000*ms*	< 2000*ms*
100	05	1	12681	12723 (0.33%)	12710 (0.22%)	12699 (0.14%)	12695 (0.11%)	12690 (0.07%)
				12775 (0.74%)	12775 (0.74%)	12698 (0.13%)	12698 (0.13%)	12692 (0.08%)
				12737 (0.44%)	12697 (0.12%)	12697 (0.12%)	12696 (0.11%)	12681 (0.00%)
100	05	2	12692	12744 (0.40%)	12726 (0.26%)	12710 (0.14%)	12703 (0.08%)	12698 (0.04%)
				12726 (0.26%)	12726 (0.26%)	12726 (0.26%)	12702 (0.07%)	12702 (0.07%)
				12779 (0.68%)	12733 (0.32%)	12733 (0.32%)	12715 (0.18%)	12703 (0.08%)
100	05	4	12810	12889 (0.61%)	12889 (0.61%)	12879 (0.53%)	12862 (0.40%)	12843 (0.25%)
				13264 (3.54%)	12852 (0.32%)	12852 (0.32%)	12852 (0.32%)	12813 (0.02%)
				12876 (0.51%)	12873 (0.49%)	12848 (0.29%)	12810 (0.00%)	12810 (0.00%)
100	05	8	12737	13030 (2.30%)	12794 (0.44%)	12785 (0.37%)	12767 (0.23%)	12754 (0.13%)
				29233 (129.51%)	12780 (0.33%)	12780 (0.33%)	12780 (0.33%)	12766 (0.22%)
				12817 (0.62%)	12817 (0.62%)	12772 (0.27%)	12772 (0.27%)	12748 (0.08%)
100	10	1	11576	11613 (0.31%)	11613 (0.31%)	11604 (0.24%)	11596 (0.17%)	11589 (0.11%)
				12868 (11.16%)	11725 (1.28%)	11725 (1.28%)	11616 (0.34%)	11616 (0.34%)
				13128 (13.40%)	11673 (0.83%)	11615 (0.33%)	11611 (0.30%)	11609 (0.28%)
100	10	2	11582	11638 (0.48%)	11638 (0.48%)	11638 (0.48%)	11624 (0.36%)	11606 (0.20%)
				11870 (2.48%)	11870 (2.48%)	11690 (0.93%)	11674 (0.79%)	11674 (0.79%)
				11643 (0.52%)	11643 (0.52%)	11643 (0.52%)	11639 (0.49%)	11619 (0.31%)
100	10	4	11654	11795 (1.20%)	11790 (1.16%)	11767 (0.96%)	11739 (0.72%)	11724 (0.60%)
				12364 (6.09%)	12364 (6.09%)	11769 (0.98%)	11769 (0.98%)	11742 (0.75%)
				13369 (14.71%)	11843 (1.62%)	11843 (1.62%)	11728 (0.63%)	11728 (0.63%)

100	10	8	11648	15822 (35.83%)	12078 (3.69%)	11866 (1.87%)	11821 (1.48%)	11778 (1.11%)
				29043 (149.33%)	13124 (12.67%)	13124 (12.67%)	11775 (1.09%)	11775 (1.09%)
				13228 (13.56%)	12886 (10.62%)	12737 (9.34%)	12122 (4.06%)	11748 (0.85%)
100	20	1	8435	8702 (3.16%)	8702 (3.16%)	8627 (2.27%)	8553 (1.39%)	8503 (0.80%)
				9153 (8.51%)	8594 (1.88%)	8594 (1.88%)	8594 (1.88%)	8594 (1.88%)
				9241 (9.55%)	8592 (1.86%)	8587 (1.80%)	8587 (1.80%)	8587 (1.80%)
100	20	2	10148	10351 (2.00%)	10351 (2.00%)	10344 (1.93%)	10302 (1.51%)	10244 (0.94%)
				12678 (24.93%)	12678 (24.93%)	12678 (24.93%)	10296 (1.45%)	10296 (1.45%)
				26235 (158.52%)	10498 (3.44%)	10371 (2.19%)	10313 (1.62%)	10313 (1.62%)
100	20	4	⌊10876	14785 (35.94%)	11213 (3.09%)	11201 (2.98%)	11141 (2.43%)	11114 (2.18%)
				14163 (30.22%)	14163 (30.22%)	14163 (30.22%)	14163 (30.22%)	12018 (10.50%)
				29263 (169.06%)	12450 (14.47%)	11550 (6.19%)	11458 (5.35%)	11424 (5.03%)
100	20	8	⌊11288	13836 (22.57%)	13836 (22.57%)	11921 (5.60%)	11837 (4.86%)	11780 (4.35%)
					12868 (13.99%)	12868 (13.99%)	12868 (13.99%)	12868 (13.99%)
				27996 (148.01%)	27996 (148.01%)	15373 (36.18%)	12330 (9.23%)	12330 (9.23%)
Guided Local Search				12/12 (8.76%)	12/12 (3.16%)	12/12 (1.45%)	12/12 (1.14%)	12/12 (0.89%)
CPLEX 12.5				11/12 (33.34%)	12/12 (7.93%)	12/12 (7.32%)	12/12 (4.29%)	12/12 (2.59%)
Gurobi 5.1				12/12 (44.13%)	12/12 (15.24%)	12/12 (4.93%)	12/12 (2.00%)	12/12 (1.65%)

⌊ The (lower) bound is not tight.

Table C.4: Solution quality over time with GLS, CPLEX and Gurobi for dataset CR-100-*

Problem i	j	k	LB	■ = Guided Local Search ■ = CPLEX 12.5 ■ = Gurobi 5.1 < 100*ms*	< 500*ms*	< 1000*ms*	< 10000*ms*	< 30000*ms*
100	05	1	1972	1993 (1.06%)	1977 (0.25%)	1975 (0.15%)	1974 (0.10%)	1973 (0.05%)
				1995 (1.16%)	1978 (0.30%)	1976 (0.20%)	1972 (0.00%)	1972 (0.00%)
				1992 (1.01%)	1975 (0.15%)	1975 (0.15%)	1972 (0.00%)	1972 (0.00%)
100	05	2	1987	2043 (2.81%)	1993 (0.30%)	1991 (0.20%)	1987 (0.00%)	1987 (0.00%)
				2010 (1.15%)	1997 (0.50%)	1997 (0.50%)	1988 (0.05%)	1987 (0.00%)
					1997 (0.50%)	1987 (0.00%)	1987 (0.00%)	1987 (0.00%)
100	05	4	1976	1998 (1.11%)	1987 (0.55%)	1983 (0.35%)	1976 (0.00%)	1976 (0.00%)
				2028 (2.63%)	1985 (0.45%)	1983 (0.35%)	1976 (0.00%)	1976 (0.00%)
				2006 (1.51%)	1978 (0.10%)	1976 (0.00%)	1976 (0.00%)	1976 (0.00%)
100	05	8	1989		2000 (0.55%)	1995 (0.30%)	1991 (0.10%)	1990 (0.05%)
					1994 (0.25%)	1994 (0.25%)	1991 (0.10%)	1989 (0.00%)
					2000 (0.55%)	1998 (0.45%)	1990 (0.05%)	1989 (0.00%)
100	10	1	1699	1776 (4.53%)	1763 (3.76%)	1745 (2.70%)	1709 (0.58%)	1701 (0.11%)
						1734 (2.06%)	1706 (0.41%)	1700 (0.05%)
						1724 (1.47%)	1701 (0.11%)	1701 (0.11%)
100	10	2	1718	1763 (2.61%)	1751 (1.92%)	1744 (1.51%)	1730 (0.69%)	1728 (0.58%)
						1766 (2.79%)	1732 (0.81%)	1726 (0.46%)
						1741 (1.33%)	1723 (0.29%)	1718 (0.00%)
100	10	4	1751	1808 (3.25%)	1794 (2.45%)	1789 (2.17%)	1761 (0.57%)	1758 (0.39%)
							1788 (2.11%)	1765 (0.79%)
							1763 (0.68%)	1753 (0.11%)

100	10	8	1694			1761	(3.95%)	1739	(2.65%)	1707	(0.76%)	1697	(0.17%)
						1730	(2.12%)	1730	(2.12%)	1718	(1.41%)	1699	(0.29%)
										1705	(0.64%)	1694	(0.00%)
100	20	1	1743	1927	(10.55%)	1897	(8.83%)	1861	(6.76%)	1819	(4.36%)	1811	(3.90%)
										1810	(3.84%)	1787	(2.52%)
												1750	(0.40%)
100	20	2	1709	1964	(14.92%)	1860	(8.83%)	1855	(8.54%)	1798	(5.20%)	1783	(4.33%)
										1838	(7.54%)	1838	(7.54%)
										1996	(16.79%)	1726	(0.99%)
100	20	4	1708			1834	(7.37%)	1834	(7.37%)	1784	(4.44%)	1770	(3.62%)
										1734	(1.52%)	1734	(1.52%)
										1757	(2.86%)	1731	(1.34%)
100	20	8	1805	2106	(16.67%)	1990	(10.24%)	1990	(10.24%)	1897	(5.09%)	1856	(2.82%)
										1924	(6.59%)	1880	(4.15%)
												1810	(0.27%)
Guided Local Search				9/12	(6.39%)	12/12	(4.08%)	12/12	(3.57%)	12/12	(1.82%)	12/12	(1.33%)
CPLEX 12.5				3/12	(1.64%)	5/12	(0.72%)	7/12	(1.18%)	12/12	(2.03%)	12/12	(1.44%)
Gurobi 5.1				2/12	(1.26%)	4/12	(0.32%)	6/12	(0.55%)	10/12	(2.13%)	12/12	(0.25%)

↓ The (lower) bound is not tight.

Table C.5: Solution quality over time with GLS, CPLEX and Gurobi for dataset DR-100-*

Problem i	j	k	LB	■ = Guided Local Search ■ = CPLEX 12.5 ■ = Gurobi 5.1 < 100*ms*	< 500*ms*	< 1000*ms*	< 10000*ms*	< 30000*ms*
100	05	1	6392	6434 (0.65%) 6507 (1.79%)	6419 (0.42%) 6411 (0.29%) 6408 (0.25%)	6415 (0.35%) 6411 (0.29%) 6408 (0.25%)	6403 (0.17%) 6405 (0.20%) 6405 (0.20%)	6401 (0.14%) 6403 (0.17%) 6402 (0.15%)
100	05	2	6396	6481 (1.32%) 6474 (1.21%)	6436 (0.62%) 6433 (0.57%) 6403 (0.10%)	6432 (0.56%) 6425 (0.45%) 6399 (0.04%)	6411 (0.23%) 6403 (0.10%) 6399 (0.04%)	6407 (0.17%) 6399 (0.04%) 6399 (0.04%)
100	05	4	6408	6553 (2.26%) 6452 (0.68%)	6462 (0.84%) 6422 (0.21%) 6421 (0.20%)	6451 (0.67%) 6422 (0.21%) 6421 (0.20%)	6434 (0.40%) 6421 (0.20%) 6419 (0.17%)	6427 (0.29%) 6421 (0.20%) 6419 (0.17%)
100	05	8	⌊6439		6563 (1.92%) 6508 (1.07%) 6527 (1.36%)	6542 (1.59%) 6508 (1.07%) 6486 (0.72%)	6488 (0.76%) 6466 (0.41%) 6481 (0.65%)	6480 (0.63%) 6466 (0.41%) 6466 (0.41%)
100	10	1	⌊6506	6960 (6.97%)	6729 (3.42%)	6688 (2.79%) 6613 (1.64%) 6628 (1.87%)	6601 (1.46%) 6537 (0.47%) 6557 (0.78%)	6585 (1.21%) 6537 (0.47%) 6549 (0.66%)
100	10	2	⌊6504	6794 (4.45%)	6723 (3.36%) 6755 (3.85%)	6693 (2.90%) 6647 (2.19%) 6738 (3.59%)	6614 (1.69%) 6608 (1.59%) 6591 (1.33%)	6597 (1.42%) 6549 (0.69%) 6582 (1.19%)
100	10	4	⌊6510	6767 (3.94%)	6763 (3.88%)	6739 (3.51%) 6661 (2.31%) 6843 (5.11%)	6653 (2.19%) 6624 (1.75%) 6584 (1.13%)	6633 (1.88%) 6612 (1.56%) 6575 (0.99%)

100	10	8	⌊6609			6900	(4.40%)	6900	(4.40%)	6801	(2.90%)	6774	(2.49%)
										6833	(3.38%)	6803	(2.93%)
										6793	(2.78%)	6760	(2.28%)
100	20	1	⌊6548	7174	(9.56%)	7174	(9.56%)	7092	(8.30%)	6897	(5.32%)	6849	(4.59%)
												6881	(5.08%)
100	20	2	⌊6428	6973	(8.47%)	6973	(8.47%)	6973	(8.47%)	6728	(4.66%)	6661	(3.62%)
										6638	(3.26%)	6638	(3.26%)
										6604	(2.73%)	6604	(2.73%)
100	20	4	⌊6467	7176	(10.96%)	7034	(8.76%)	7033	(8.75%)	6846	(5.86%)	6800	(5.14%)
												6941	(7.32%)
												6834	(5.67%)
100	20	8	⌊6590			7526	(14.20%)	7323	(11.12%)	7153	(8.54%)	7093	(7.63%)
												7141	(8.36%)
												7103	(7.78%)
Guided Local Search				9/12	(5.39%)	12/12	(4.98%)	12/12	(4.45%)	12/12	(2.84%)	12/12	(2.43%)
CPLEX 12.5				3/12	(1.22%)	5/12	(1.19%)	7/12	(1.16%)	9/12	(1.26%)	11/12	(2.31%)
Gurobi 5.1				0/12	(0.00%)	4/12	(0.47%)	7/12	(1.68%)	9/12	(1.09%)	12/12	(2.26%)

⌊ The (lower) bound is not tight.

Table C.6: Solution quality over time with GLS, CPLEX and Gurobi for dataset ER-100-*

Problem i	j	k	LB	■ = Guided Local Search ■ = CPLEX 12.5 ■ = Gurobi 5.1 < 100*ms*	< 500*ms*	< 1000*ms*	< 10000*ms*	< 30000*ms*
100	05	1	12727	12764 (0.29%)	12764 (0.29%)	12752 (0.19%)	12735 (0.06%)	12734 (0.05%)
				12795 (0.53%)	12795 (0.53%)	12748 (0.16%)	12744 (0.13%)	12731 (0.03%)
				13053 (2.56%)	12819 (0.72%)	12802 (0.58%)	12730 (0.02%)	12728 (0.00%)
100	05	2	12732	12761 (0.22%)	12760 (0.21%)	12749 (0.13%)	12732 (0.00%)	12732 (0.00%)
				12892 (1.25%)	12808 (0.59%)	12794 (0.48%)	12749 (0.13%)	12749 (0.13%)
					12755 (0.18%)	12755 (0.18%)	12732 (0.00%)	12732 (0.00%)
100	05	4	12849	13442 (4.61%)	12951 (0.79%)	12923 (0.57%)	12866 (0.13%)	12866 (0.13%)
					12910 (0.47%)	12885 (0.28%)	12885 (0.28%)	12860 (0.08%)
					12907 (0.45%)	12860 (0.08%)	12849 (0.00%)	12849 (0.00%)
100	05	8	12775		13106 (2.59%)	12965 (1.48%)	12804 (0.22%)	12791 (0.12%)
				13287 (4.00%)	12804 (0.22%)	12804 (0.22%)	12775 (0.00%)	12775 (0.00%)
					12884 (0.85%)	12833 (0.45%)	12786 (0.08%)	12785 (0.07%)
100	10	1	12091	12360 (2.22%)	12360 (2.22%)	12260 (1.39%)	12127 (0.29%)	12102 (0.09%)
					12332 (1.99%)	12310 (1.81%)	12160 (0.57%)	12151 (0.49%)
						12405 (2.59%)	12157 (0.54%)	12091 (0.00%)
100	10	2	11967	13726 (14.69%)	12138 (1.42%)	12089 (1.01%)	11990 (0.19%)	11983 (0.13%)
					12902 (7.81%)	12853 (7.40%)	12028 (0.50%)	11996 (0.24%)
						12177 (1.75%)	12020 (0.44%)	11973 (0.05%)
100	10	4	12112		12394 (2.32%)	12306 (1.60%)	12177 (0.53%)	12167 (0.45%)
						12500 (3.20%)	12279 (1.37%)	12193 (0.66%)
						13279 (9.63%)	12232 (0.99%)	12182 (0.57%)

100	10	8	⌊12066			12347	(2.32%)	12347	(2.32%)	12184	(0.97%)	12139	(0.60%)
								12384	(2.63%)	12164	(0.81%)	12164	(0.81%)
										12201	(1.11%)	12170	(0.86%)
100	20	1	8941	11408	(27.59%)	9238	(3.32%)	9220	(3.12%)	9060	(1.33%)	9026	(0.95%)
								9362	(4.70%)	9258	(3.54%)	9095	(1.72%)
										9137	(2.19%)	9137	(2.19%)
100	20	2	10973			11529	(5.06%)	11331	(3.26%)	11112	(1.26%)	11053	(0.72%)
										11399	(3.88%)	11255	(2.56%)
										11252	(2.54%)	11064	(0.82%)
100	20	4	⌊11749			13477	(14.70%)	12395	(5.49%)	12144	(3.36%)	12103	(3.01%)
												13162	(12.02%)
												12422	(5.72%)
100	20	8	⌊12028					13142	(9.26%)	12698	(5.57%)	12615	(4.88%)
										14101	(17.23%)	13654	(13.51%)
												16320	(35.68%)
Guided Local Search				6/12	(8.27%)	11/12	(3.20%)	12/12	(2.48%)	12/12	(1.15%)	12/12	(0.92%)
CPLEX 12.5				3/12	(1.92%)	6/12	(1.93%)	9/12	(2.32%)	11/12	(2.58%)	12/12	(2.68%)
Gurobi 5.1				1/12	(2.56%)	4/12	(0.55%)	7/12	(2.18%)	10/12	(0.79%)	12/12	(3.83%)

⌊ The (lower) bound is not tight.

Acronyms

4S	Smart Chips For Smart Surroundings.
API	Application Programming Interface.
ASIC	Application Specific Integrated Circuit.
BFS	Breadth-first Search.
BIST	Built-in Self Test.
BPP	Bandwidth Packing Problem.
CAES	Computer Architecture For Embedded Systems.
CMOS	Complementary Metal-oxide-semiconductor.
CPU	Central Processing Unit.
CRISP	Cutting Edge Reconfigurable ICs For Stream Processing.
DDR	Double Data Rate.
DLI	Die Link Interface.
DM	Dependability Manager.
DRAM	Dynamic Random-access Memory.
DSP	Digital Signal Processor.
FFT	Fast Fourier Transform.
FIFO	First-in First-out.
FLIT	Flow Control Digit.
FP6	Sixth Framework Programme.
FP7	Seventh Framework Programme.
FPGA	Field Programmable Gate Array.
GAP	Generalized Assignment Problem.
GLS	Guided Local Search.
GNSS	Global Navigation Satellite System.
GPD	General Purpose Device.
GPP	General Purpose Processor.
GPS	Global Positioning System.
GQAP	Generalized Quadratic Assignment Problem.
GSP	General Stream Processor.
HWA	Hardware Accelerator.

I/O	Input / Output.
IC	Integrated Circuit.
ILP	Integer Linear Program.
MCM	Maximum Cycle Mean.
MCP	Multi-channel Port.
MMKP	Multi-dimensional Multiple-choice Knapsack Problem.
MPSoC	Multi-processor System-on-Chip.
MRGAP	Multi-resource Generalized Assignment Problem.
MRQAP	Multi-resource Quadratic Assignment Problem.
MRQARP	Multi-resource Quadratic Assignment And Routing Problem.
NoC	Network-on-chip.
PCB	Printed Circuit Board.
PE	Processing Element.
PVT	Position, Velocity And Time.
QoS	Quality Of Service.
RF	Radio Frequency.
RFD	Reconfigurable Fabric Device.
SCPP	Shortest Capacitated Path Problem.
SMP	Symmetric Multiprocessing.
SoC	System-on-chip.
SRAM	Static Random-access Memory.
STARS	Sensor Technology Applied In Reconfigurable Systems.
UFP	Unsplittable Flow Problem.
VLIW	Very Large Instruction Word.
VT	Virtual Tile.

Bibliography

[1] Luca Abeni and Giorgio Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, July 2004. ISSN 0922-6443. doi: 10.1023/B:TIME.0000027934.77900.22.

[2] Ellen Allen, Richard Helgason, Jeffrey Kennington, and Bala Shetty. A generalization of Polyak's convergence result for subgradient optimization. *Mathematical Programming*, 37(3):309–317, May 1987. ISSN 0025-5610. doi: 10.1007/BF02591740.

[3] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. doi: 10.1145/1465482.1465560.

[4] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. doi: 10.1145/1465482.1465560.

[5] Ali Amiri and Reza Barkhi. The combinatorial bandwidth packing problem. *European Journal of Operational Research*, 208(1):37 – 45, 2011. ISSN 0377-2217. doi: 10.1016/j.ejor.2010.08.005.

[6] Atmel. AT91CAP9S250A customizable microcontroller processor, March 2009. http://www.atmel.com/dyn/resources/prod_documents/doc6264.pdf [Online; accessed January 3, 2011].

[7] Baruch Awerbuch, Yossi Azar, and Amir Epstein. The price of routing unsplittable flow. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 57–66, New York, NY, USA, 2005. ACM. ISBN 1-58113-960-8. doi: 10.1145/1060590.1060599.

[8] Yossi Azar and Oded Regev. Strongly polynomial algorithms for the unsplittable flow problem. In *Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 15–29, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42225-0. doi: 10.1007/3-540-45535-3_2.

[9] Jonathan F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*. Nonconvex Optimization and Its Applications. Springer, 1998. ISBN 978-07-923-5458-1. doi: 10.1007/978-1-4757-2836-1.

[10] J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. doi: 10.1057/jors.1990.166.

[11] J. E. Beasly. OR-Library: a collection of test data sets for a variety of Operations Research (OR) problems. http://people.brunel.ac.uk/~mastjjb/jeb/info.html, June 2012. [Online; accessed November 14, 2012].

[12] James Beck and Daniel Siewiorek. Modeling multicomputer task allocation as a vector packing problem. In *ISSS '96: Proceedings of the 9th international symposium on System synthesis*, pages 115–120, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7563-2. doi: 10.1109/ISSS.1996.565892.

[13] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, May 2011. ISSN 0001-0782. doi: 10.1145/1941487.1941507.

[14] D. Braess. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, 12 (1):258–268, 1968. ISSN 1432-5217. doi: 10.1007/BF01918335.

[15] P.M. Camerini, L. Fratta, and F. Maffioli. On improving relaxation methods by modified gradient techniques. *Mathematical Programming Studies*, 3(4):26–34, 1975. doi: 10.1007/BFb0120697.

[16] Alvaro E. Campos and Dionel A. Suazo. Data-race and concurrent-write freedom are undecidable. *Comput. Lang. Syst. Struct.*, 29(1-2):1–13, April 2003. ISSN 1477-8424. doi: 10.1016/S1477-8424(03)00014-9.

[17] Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 213–222, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. ISBN 0-89871-453-2.

[18] Andrew A. Chien, Allan Snavely, and Mark Gahagan. 10x10: A general-purpose architectural approach to heterogeneity and energy efficiency. *Procedia Computer Science*, 4(0):1987 – 1996, 2011. ISSN 1877-0509. doi: 10.1016/j.procs.2011.04.217. Proceedings of the International Conference on Computational Science, ICCS 2011.

[19] Chen-Ling Chou and R. Marculescu. FARM: Fault-aware resource management in NoC-based multiprocessor platforms. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2011), Grenoble*, pages 1–6. European Design and Automation Association, March 2011. doi: 10.1109/DATE.2011.5763113.

[20] Chen-Ling Chou and Radu Marculescu. Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 161–166, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-824-4. doi: 10.1145/1289816.1289857.

[21] Reuven Cohen, Liran Katzir, and Danny Raz. An efficient approximation for the generalized assignment problem. *Inf. Process. Lett.*, 100(4):162–166, 2006. ISSN 0020-0190. doi: 10.1016/j.ipl.2006.06.003.

[22] Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. Pricing networks with selfish routing. In *Proceeding of the 35th Symposium on Theory of Computing (STOC)*, pages 521–530, 2003.

[23] Katherine Compoton and Scott Hauck. An introduction to reconfigurable computing. Technical report, Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL USA, 1999.

[24] Marie-Christine Costa, Alain Hertz, and Michel Mittaz. Bounds and heuristics for the shortest capacitated paths problem. *Journal of Heuristics*, 8(4):449–465, July 2002. ISSN 1381-1231. doi: 10.1023/A:1015492014030.

[25] M. Cuviello, S. Dey, Xiaoliang Bai, and Yi Zhao. Fault modeling and simulation for crosstalk in system-on-chip interconnects. In *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, pages 297–303, 1999. doi: 10.1109/ICCAD.1999.810665.

[26] William Dally. The path to exascale, November 2014. http://on-demand.gputechconf.com/supercomputing/2014/presentation/SC415-nvidia-path-exascale.pdf [Online; accesses November 11, 2016].

[27] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, October 1974. ISSN 0018-9200. doi: 10.1023/A:1008373903657.

[28] Robert P. Dick, David L. Rhodes, and Wayne Wolf. TGFF: task graphs for free. In *CODES/CASHE '98: Proc. of the 6th international workshop on Hardware/software codesign*, pages 97–101, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8442-9.

[29] J. Dongarra and A. Lastovetsky. An overview of heterogeneous high performance and grid computing. *Engineering the Grid: Status and Perspective*, February 2006 2006.

[30] N. P. Edwards. On the architectural requirements of an engineered system. Technical report, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598, August 1977. Research Report RC 6688, IBM Thomas J. Watson Research Center.

[31] Mohammad Abdullah Al Faruque, Rudolf Krist, and Jörg Henkel. ADAM: run-time agent-based distributed application mapping for on-chip communication. In *DAC '08: Proc. of the 45th annual conference on Design automation*, pages 760–765, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-115-6. doi: 10.1145/1391469.1391664.

[32] Marshall L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Manage. Sci.*, 50(12 Supplement):1861–1871, December 2004. ISSN 0025-1909. doi: 10.1287/mnsc.1040.0263.

[33] Christodoulos A. Floudas and Panos M. Pardalos, editors. *Encyclopedia of Optimization, Second Edition*. Springer, 2009. ISBN 978-0-387-74758-3.

[34] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975. ISSN 00975397. doi: 10.1137/0204035.

[35] M. R. Garey and D. S. Johnson. "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *J. ACM*, 25:499–508, July 1978. ISSN 0004-5411. doi: 10.1145/322077.322090.

[36] Bezalel Gavish and Hasan Pirkul. Algorithms for the multi-resource generalized assignment problem. *Manage. Sci.*, 37(6):695–713, 1991. ISSN 0025-1909. doi: 10.1287/mnsc.37.6.695.

[37] M.L. Gavrilova, Y. Wang, C.J.K. Tan, Y.Y. Yao, and G. Wang. *Transactions on Computational Science II*. Number v. 2 in LNCS sublibrary. SL 1, Theoretical computer science and general issues. Springer, 2008. ISBN 9783540875628. URL http://books.google.nl/books?id=mXo1GzcUVI8C.

[38] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986. ISSN 0305-0548. doi: 10.1016/0305-0548(86)90048-1. Applications of Integer Programming.

[39] Fred Glover, Darwin Klingman, and Nancy V. Phillips. *Network models in optimization and their applications in practice*. Wiley, 1992. ISBN 978-0-471-57138-4.

[40] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *CONTROL AND CYBERNETICS*, 39:653–684, 2000.

[41] Andrew V. Goldberg, Jeffrey D. Oldham, Serge Plotkin, and Cliff Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In Robert E. Bixby, E. Andrew Boyd, and Roger Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 338–352. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64590-0. doi: 10.1007/3-540-69346-7_26.

[42] Dimitry Gromov. *Analysis of Hierarchical Structures for Hybrid Control Systems*. PhD thesis, Technical University of Berlin, Germany, Sep 2010.

[43] Vishal Gupta and Karsten Schwan. Brawny vs. wimpy: Evaluation and analysis of modern workloads on heterogeneous processors. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing Workshop (IPDPSW)*, pages 74–83. IEEE, May 2013. doi: 10.1109/IPDPSW.2013.130.

[44] Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual. http://www.gurobi.com, 2013. [Online; accessed April 12, 2013].

[45] John L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5): 532–533, May 1988. ISSN 0001-0782. doi: 10.1145/42411.42415.

[46] Andreas Hansson, Kees Goossens, and Andrei Rădulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 75–80, New York, NY, USA, 2005. ACM. ISBN 1-59593-161-9. doi: 10.1145/1084834.1084857.

[47] Andreas Hansson, Martijn Coenen, and Kees Goossens. Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip. In *DATE '07: Proc. of the conference on Design, automation and test in Europe*, pages 954–959, San Jose, CA, USA, 2007. EDA Cons. ISBN 978-3-9810801-2-4.

[48] Tim Harris, Simon Marlow, Simon Peyton-Jones, and Maurice Herlihy. Composable memory transactions. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, pages 48–60, New York, NY, USA, 2005. ACM. ISBN 1-59593-080-9. doi: 10.1145/1065944.1065952.

[49] Hideki Hashimoto and Mutsunori Yagiura. A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In Jano Hemert and Carlos Cotta, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 4972 of *Lecture Notes in Computer Science*, pages 254–265. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-78603-0. doi: 10.1007/978-3-540-78604-7_22.

[50] Michael Held, Philip Wolfe, and Harlan P. Crowder. Validation of subgradient optimization. *Math. Program.*, 6(1):62–88, 12 1974. doi: 10.1007/BF01580223.

[51] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January 2000. ISSN 1077-2626. doi: 10.1109/2945.841119.

[52] Marcos Herve, Erika Cota, Fernanda Lima Kastensmidt, and Marcelo Lubaszewski. Diagnosis of interconnect shorts in mesh NoCs. *Networks-on-Chip, International Symposium on*, pages 256–265, 2009. doi: 10.1109/NOCS.2009.5071475.

[53] Paul M. Heysters, Gerard J. M. Smit, and Egbert Molenkamp. Energy-efficiency of the MONTIUM reconfigurable tile processor. In Toomas P. Plaks, editor, *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'04, June 21-24, 2004, Las Vegas, Nevada, USA*, pages 38–44. CSREA Press, 2004.

[54] C. A. R. Hoare. Communicating sequential processes. *Communications of ACM*, 21 (8):666–677, August 1978. ISSN 0001-0782. doi: 10.1145/359576.359585.

[55] P. K. F. Hölzenspies. *On run-time exploitation of concurrency*. PhD thesis, University of Twente, Enschede, The Netherlands, April 2010. doi:doi: 10.3990/1.9789036530217.

[56] P. K. F. Hölzenspies, J. Kuper, G. J. M. Smit, and J. L. Hurink. Demonstration of run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (MPSoC). In B. R. H. M. Haverkort, J. P. Katoen, and L. Thiele, editors, *Dagstuhl Seminar Proceedings 07101, Dagstuhl Wadern, Germany*, volume 07101, Dagstuhl, Germany, October 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI).

[57] Philip K. F. Hölzenspies, Johann L. Hurink, Jan Kuper, and Gerard J. M. Smit. Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip. In *DATE '08: Proc. of the conference on Design, automation and test in Europe*, pages 212–217, March 2008. doi: 10.1109/DATE.2008.4484688.

[58] M. Hosseinabady and J.L. Nunez-Yanez. Run-time resource management in fault-tolerant network on reconfigurable chips. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 574–577, aug. 2009. doi: 10.1109/FPL.2009.5272400.

[59] Heikki Hurskainen, Jussi Raasakka, Tapani Ahonen, and Jari Nurmi. Multicore software-defined radio architecture for GNSS receiver signal processing. *EURASIP J. Embedded Syst.*, 2009:3:1–3:10, January 2009. ISSN 1687-3955. doi: 10.1155/2009/543720.

[60] IBM. IBM ILOG CPLEX: High-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming. http://www.ibm.com/software/integration/optimization/cplex-optimization-studio/, 2013. [Online; accessed April 12, 2013].

[61] ICT-2007.3.4. Architecture Paradigms and Programming Languages for Efficient programming of multiple CORES (Apple-CORE) project. http://cordis.europa.eu/project/rcn/85401_en.html, 2007. ICT-2007.3.4 - Computing systems.

[62] ICT-2009.3.6. Asynchronous and Dynamic Virtualisation through performance ANalysis to support Concurrency Engineering (ADVANCE) project. http://cordis.europa.eu/project/rcn/93753_en.html, 2010. ICT-2009.3.6 - Computing Systems.

[63] ICT-2009.8.1. Service-oriented Operating Systems. http://cordis.europa.eu/project/rcn/92807_en.html, 2010. ICT-2009.8.1 - FET proactive 1: Concurrent Tera-device Computing.

[64] ICT-2011.9.8. Self Energy-Supporting Autonomous Computation. http://cordis.europa.eu/project/rcn/105544_en.html, 2012. ICT-2011.9.8 - FET Proactive: Minimising Energy Consumption of Computing to the Limit (MINECC).

[65] ICT-2013.3.4. Programming Large Scale Heterogeneous Infrastructures (POLCA) project. http://cordis.europa.eu/project/rcn/109259_en.html, 2013. ICT-2013.3.4 - Advanced computing, embedded and control systems.

[66] Mathieu Jacomy, Sébastian Heymann, Tommaso Venturini, and Mathieu Bastian. Forceatlas2, a graph layout algorithm for handy network visualization, 2011. http://webatlas.fr/tempshare/ForceAtlas2_Paper.pdf [Online; accessed January 12, 2016].

[67] Dick James. Intel's 14nm Parts are Finally Here! Technical report, Chipworks, October 2014. http://www.chipworks.com/about-chipworks/overview/blog/intel's-14-nm-parts-are-finally-here [Online; accessed August 15, 2016].

[68] Chris R. Jesshope. A model for the design and programming of multicores. In Lucio Grandinetti, editor, *High Performance Computing Workshop*, volume 16 of *Advances in Parallel Computing*, pages 37–55. IOS Press, 2008. ISBN 978-1-58603-839-7.

[69] D. I. Kang, J. Suh, O. McMahon, and S. Crago. Preliminary study towards intelligent run-time resource management techniques for large multi-core architectures. Technical report, University of Southern California – Information Sciences Institute, September 2007. High-Performance Computing Workshop.

[70] N. K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. PhD thesis, University of Twente, Enschede, January 2007.

[71] Nikolay Kavaldjiev, Gerard J.M. Smit, Pascal T. Wolkotte, and Pierre G. Jansen. Providing QoS Guarantees in a NoC by Virtual Channel Reservation. In Koen Bertels, JoãoM.P. Cardoso, and Stamatis Vassiliadis, editors, *Reconfigurable Computing: Architectures and Applications*, volume 3985 of *Lecture Notes in Computer Science*, pages 299–310. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-36708-6. doi: 10.1007/11802839_38.

[72] H. G. Kerkhoff and X. Zhang. Design of an infrastructural ip dependability manager for a dependable reconfigurable many-core processor. In *Fifth IEEE International Symposium on Electronic Design, Test & Applications, DELTA 2010, Ho Chi Minh City, Vietnam*, pages 270–275, Los Alamitos, CA, January 2010. IEEE Computer Society Press. doi: 10.1109/DELTA.2010.57.

[73] Angeliki Kritikakou, Francky Catthoor, Vasilios Kelefouras, and Costas Goutis. A systematic approach to classify design-time global scheduling techniques. *ACM Comput. Surv.*, 45(2):14:1–14:30, March 2013. ISSN 0360-0300. doi: 10.1145/2431211.2431213.

[74] A. Kumar, B. Mesman, B. Theelen, H. Corporaal, and H. Yajun. Resource manager for non-preemptive heterogeneous multiprocessor system-on-chip. In *ESTMED '06: Proceedings of the 2006 IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia*, pages 33–38, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7803-9783-5. doi: 10.1109/ESTMED.2006.321271.

[75] Rakesh Kumar, Keith I. Farkas, Parthasarathy Ranganathan, and Dean M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proc. of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, pages 81–92, Washington, DC, USA, 2003. IEEE Computer Society. doi: 10.1109/MICRO.2003.1253185.

[76] Anany Levitin. Do we teach the right algorithm design techniques? *SIGCSE Bull.*, 31 (1):179–183, 1999. ISSN 0097-8418. doi: 10.1145/384266.299747.

[77] Keqin Li and Kam Hoi Cheng. A two dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. In *CSC '90: Proceedings of the 1990 ACM annual conference on Cooperation*, pages 22–27, New York, NY, USA, 1990. ACM. ISBN 0-89791-348-5. doi: 10.1145/100348.100352.

[78] T. Marescaux, J y. Mignolet, A. Bartic, W. Moffat, D. Verkest, and S. Vernalde. Networks on chip as hardware components of an OS for reconfigurable systems. In *In Proceedings of 13th International Conference on Field Programmable Logic and Applications*, pages 595–605, 2003.

[79] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990. ISBN 0-471-92420-2.

[80] Mark McKeown. FFT Implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs. Technical report, Texas Instruments, June 2010.

[81] Orlando Moreira, Jacob Jan-David Mol, and Marco J. G. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *SAC '07: Proc. of the 2007 ACM symposium on Applied computing*, pages 1557–1564, New York, NY, USA, 2007. ACM. ISBN 1-59593-480-4. doi: 10.1145/1244002.1244335.

[82] Tin-Fook Ngai. Dynamic resource allocation in a hierarchical multiprocessor system: a preliminary study. Technical Report CSL-TR-86-3 10, Computer Systems Laboratory, Stanford University, Stanford, CA, USA, October 1986. URL http://books.google.nl/books?id=GKWCPwAACAAJ.

[83] V. Nollet, T. Marescaux, P. Avasare, and J-Y. Mignolet. Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 234–239, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2288-2. doi: 10.1109/DATE.2005.91.

[84] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J-Y. Mignolet. Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 234–239, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2288-2. doi: 10.1109/DATE.2005.91.

[85] Vincent Nollet. *Run-time management for future MPSoC platforms*. PhD thesis, Technische Universiteit Eindhoven, 2008.

[86] Vincent Nollet, Théodore Marescaux, Diederik Verkest, Jean-Yves Mignolet, and Serge Vernalde. Operating-system controlled network on chip. In *Proceedings of the 41st annual Design Automation Conference*, DAC '04, pages 256–259, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-8. doi: 10.1145/996566.996637.

[87] NVIDIA Corporation. Tesla C2050 Performance Benchmarks. http://www.siliconmechanics.com/files/C2050Benchmarks.pdf, 2010.

[88] Joel Oughton. Improved network map display. Undergraduate honours thesis, University of Waikato, Waikato, New Zealand, October 2011.

[89] Ken Perlin and David Fox. Pad: an alternative approach to the computer interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 57–64, New York, NY, USA, 1993. ACM. ISBN 0-89791-601-8. doi: 10.1145/166117.166125.

[90] Teemu Pitkänen, Risto Mäkinen, Jari Heikkinen, Tero Partanen, and Jarmo Takala. Low-power, high-performance TTA processor for 1024-point fast fourier transform. In Stamatis Vassiliadis, Stephan Wong, and Timo Hämäläinen, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation, 6th International Workshop, SAMOS 2006, Samos, Greece, July 17-20, 2006, Proceedings*, volume 4017 of *Lecture Notes in Computer Science*, pages 227–236. Springer, 2006. doi: 10.1007/11796435_24.

[91] Recore Systems. Reconfigurable computing technology, January 2011. URL http://www.recoresystems.com/technology/.

[92] Recore Systems BV. Xentium® VLIW DSP IP core. http://www.recoresystems.com/fileadmin/downloads/Product_briefs/2012-2.0_Xentium_Product_Brief.pdf, 2012.

[93] J. M. Robson. Bounds for some functions concerning dynamic storage allocation. *J. ACM*, 21(3):491–499, Jul 1974. ISSN 0004-5411. doi: 10.1145/321832.321846.

[94] D. Romero Morales and H. Edwin Romeijn. The generalized assignment problem and extensions. In Ding-Zhu. Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume B, pages 259–311. Springer US, 2005. ISBN 978-0-387-23829-6. doi: 10.1007/0-387-23830-1_6.

[95] G. Terry Ross and Andris A. Zoltners. Weighted assignment models and their application. *Management Science*, 25(7):pp. 683–696, Juli 1979. ISSN 00251909. doi: 10.1287/mnsc.25.7.683.

[96] Jochem H. Rutgers. *Programming models for many-core architectures: a co-design approach*. PhD thesis, University of Twente, Enschede, May 2014.

[97] Nissim Saban. Multicore DSP vs GPUs, 2011. Texas Instruments, http://www.sagivtech.com/contentManagment/uploadedFiles/fileGallery/Multi_core_DSPs_vs_GPUs_TI_for_distribution.pdf [Online; accessed November 11, 2016].

[98] Vivek Sarkar, William Harrod, and Allan E Snavely. Software challenges in extreme scale systems. *Journal of Physics: Conference Series*, 180(1), 2009. doi: 10.1088/1742-6596/180/1/012045.

[99] E. Schüler, R. König, J. Becker, G. K. Rauwerda, M. D. van de Burgwal, and G. J. M. Smit. Smart chips for smart surroundings – 4s. In J. M. P. Cardoso and M. Hübner, editors, *Reconfigurable Computing: From Fpgas to Hardware/Software Codesign*, pages 117–148. Springer Verlag, London, 2011. doi: 10.1007/978-1-4614-0061-5_6.

[100] Mingoo Seok, Dongsuk Jeon, Chaitali Chakrabarti, David Blaauw, and Dennis Sylvester. A 0.27v 30mhz 17.7nj/transform 1024-pt complex FFT core with super-pipelining. In *IEEE International Solid-State Circuits Conference, ISSCC 2011, Digest of Technical Papers, San Francisco, CA, USA, 20-24 February, 2011*, pages 342–344. IEEE, 2011. ISBN 978-1-61284-303-2. doi: 10.1109/ISSCC.2011.5746346.

[101] Hamid Shojaei, AmirHossein Ghamarian, Twan Basten, Marc Geilen, Sander Stuijk, and Rob Hoes. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 917–922, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-497-3. doi: 10.1145/1629911.1630147.

[102] Hamid Shojaei, Twan Basten, Marc Geilen, and Azadeh Davoodi. A fast and scalable multidimensional multiple-choice knapsack heuristic. *ACM Trans. Des. Autom. Electron. Syst.*, 18(4):51:1–51:32, October 2013. ISSN 1084-4309. doi: 10.1145/2541012.2541014.

[103] A.K. Singh, Wu Jigang, A. Prakash, and T. Srikanthan. Efficient heuristics for minimizing communication overhead in NoC-based heterogeneous MPSoC platforms. In *Rapid System Prototyping, 2009. RSP '09. IEEE/IFIP International Symposium on*, pages 55–60, June 2009. doi: 10.1109/RSP.2009.18.

[104] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1:1–1:10, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2071-9. doi: 10.1145/2463209.2488734.

[105] Robert Spence. *Information Visualization: Design for Interaction (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2007. ISBN 0132065509.

[106] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., New York, NY, USA, 1st edition, 2000. ISBN 0824793188.

[107] Liesbeth Steffens, Gerhard Fohler, Giuseppe Lipari, and Giorgio Buttazzo. Resource reservation in real-time operating systems – a joint industrial and academic position. In *Proceedings of the International Workshop on Advanced Real-Time Operating System Services (ARTOSS)*, pages 25 – 30, July 2003.

[108] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application*. Krieger Publishing Company, January 1986.

[109] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Trans. Netw.*, 6(5):611–624, October 1998. ISSN 1063-6692. doi: 10.1109/90.731196.

[110] T. Stoilov and K. Stoilova. Non-iterative co-ordination in multilevel systems. *International Journal of Systems Science*, 29(12):1393–1416, 1998. doi: 10.1080/00207729808929625.

[111] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *DAC '07: Proc. of the 44th annual Design Automation Conference*, pages 777–782, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-627-1. doi: 10.1145/1278480.1278674.

[112] Hermen A. Toersche. On chip network support in Linux for a General Stream Processor. Master's thesis, University of Twente, December 2010.

[113] Edward P. K. Tsang. Scheduling techniques – a comparative study. *British Telecom Technology Journal*, 13(1):16 –28, 1995.

[114] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics*. MIT Press, Cambridge, MA, USA, 1990. ISBN 0-444-88074-7.

[115] John G. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):767–768, October 1952. doi: 10.1680/ipeds.1952.11362.

[116] M. H. Wiggers. *Aperiodic Multiprocessor Scheduling for Real-Time Stream Processing Applications*. PhD thesis, University of Twente, June 2009.

[117] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *SCOPES '07: Proceedingsof the 10th international workshop on Software & compilers for embedded systems*, pages 11–22, New York, NY, USA, 2007. ACM. doi: 10.1145/1269843.1269846.

[118] Pascal. T. Wolkotte. *Exploration within the Network-on-Chip Paradigm*. PhD thesis, University of Twente, Enschede, January 2009.

[119] Yanghua Xiao, Wentao Wu, Jian Pei, Wei Wang, and Zhenying He. Efficiently indexing shortest paths by exploiting symmetry in graphs. In Martin L. Kersten, Boris Novikov, Jens Teubner, Vladimir Polutin, and Stefan Manegold, editors, *EDBT*, volume 360 of *ACM International Conference Proceeding Series*, pages 493–504. ACM, 2009. ISBN 978-1-60558-422-5. doi: 10.1145/1516360.1516418.

[120] Mutsunori Yagiura. MRGAP (multi-resource generalized assignment problem) instances. http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/mrgap/, 2004. [Online; accessed April 12, 2013].

[121] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. A path relinking approach for the generalized assignment problem. In *Proc. International Symposium on Scheduling*, pages 105–108, 2002.

[122] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2): 133–151, 2004. doi: 10.1287/ijoc.1030.0036.

[123] Mutsunori Yagiura, Shinji Iwasaki, Toshihide Ibaraki, and Fred Glover. A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discret. Optim.*, 1(1):87–98, June 2004. ISSN 1572-5286. doi: 10.1016/j.disopt.2004.03.005.

[124] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169(2):548–569, 2006. ISSN 0377-2217. doi: 10.1016/j.ejor.2004.08.015. Feature Cluster on Scatter Search Methods for Optimization.

[125] Mutsunori Yagiura, Akira Komiya, Kenya Kojima, Koji Nonobe, Hiroshi Nagamochi, Toshihide Ibaraki, and Fred Glover. A path relinking approach for the multi-resource generalized quadratic assignment problem. In Thomas Stützle, Mauro Birattari, and Holger H. Hoos, editors, *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007, Proceedings*, volume 4638 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2007. ISBN 978-3-540-74445-0. doi: 10.1007/978-3-540-74446-7_9.

[126] Edward Z. Yang, 2010. http://blog.ezyang.com/2010/07/graphs-not-grids/ [Online; accessed November 8, 2016].

[127] Ch. Ykman-Couvreur, V. Nollet, Fr. Catthoor, and H. Corporaal. Fast multi-dimension multi-choice knapsack heuristic for MP-soC run-time management. *System-on-Chip, 2006. International Symposium on*, pages 1–4, November 2006. doi: 10.1109/ISSOC.2006.321966.

[128] Ch. Ykman-Couvreur, V. Nollet, Th. Marescaux, E. Brockmeyer, Fr. Catthoor, and H. Corporaal. Design-time application mapping and platform exploration for MP-soC customised run-time management. *Computers & Digital Techniques, IET*, 1(2): 120–128, March 2007. ISSN 1751-8601. doi: 10.1049/iet-cdt:20060031.

[129] X. Zhang. *Towards a dependable homogeneous many-processor system-on-chip*. PhD thesis, University of Twente, Enschede, October 2014. doi: 10.3990/1.9789036537728.

[130] X. Zhang, H. G. Kerkhoff, and B. Vermeulen. On-chip scan-based test strategy for a dependable many-core processor using a NoC as a test access mechanism. In *Proceedings of the 13th Euromicro Conference on Digital System Design, DSD 2010, Lille, France*, pages 531–537, Los Alamitos, September 2010. IEEE Computer Society. doi: 10.1109/DSD.2010.16.

[131] Yutian Zhao, Ahmet T. Erdogan, and Tughrul Arslan. A low-power and domain-specific reconfigurable FFT fabric for system-on-chip applications. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CO, USA*. IEEE Computer Society, 2005. doi: 10.1109/IPDPS.2005.39.

[132] Rong Zhou and Eric A. Hansen. Beam-stack search: Integrating backtracking with beam search. In Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors, *ICAPS*, pages 90–98. AAAI, 2005. ISBN 1-57735-220-3.

[133] Mark Ziegelmann. *Constrained Shortest Paths and Related Problems - Constrained Network Optimization*. VDM Verlag, Saarbrücken, Germany, Germany, 2007. ISBN 978-3-836-44633-4.

[134] Peter Zipf, Gilles Sassatelli, Nurten Utlu, Nicolas Saint-Jean, Pascal Benoit, and Manfred Glesner. A decentralised task mapping approach for homogeneous multiprocessor network-on-chips. *Int. J. Reconfig. Comput.*, 2009:1–14, 2009. ISSN 1687-7195. doi: 10.1155/2009/453970.

List of Publications

[TDtB:1] Timon D. ter Braak. Run-time spatial resource management in heterogeneous MPSoCs. Master's thesis, University of Twente, August 2009.

[TDtB:2] Philip K. F. Hölzenspies, Timon D. ter Braak, Jan Kuper, Gerard J. M. Smit, and Johann L. Hurink. Run-time spatial mapping of streaming applications to heterogeneous multi-processor systems. *International Journal of Parallel Programming*, 38(1):68–83, November 2009. ISSN 0885-7458. doi: 10.1007/s10766-009-0120-y.

[TDtB:3] Timon D. ter Braak, Philip K. F. Hölzenspies, Jan Kuper, Johann L. Hurink, and Gerard J. M. Smit. Run-time spatial resource management for real-time applications on heterogeneous MPSoCs. In Giovanni De Micheli, Bashir M. Al-Hashimi, Wolfgang Müller, and Enrico Macii, editors, *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2010)*, pages 357–362. European Design and Automation Association, March 2010. doi: 10.1109/DATE.2010.5457177.

[TDtB:4] Timon D. ter Braak, Stephen T. Burgess, Heiki Hurskainen, Hans G. Kerkhoff, Bart Vermeulen, and Xiao Zhang. On-line dependability enhancement of multiprocessor SoCs by resource management. In *Proceedings of the 2010 International Symposium on System-on-Chip, Tampere, Finland*, pages 103–110, Piscataway, September 2010. IEEE Circuits and Systems Society. doi: 10.1109/ISSOC.2010.5625564.

[TDtB:5] Timon D. ter Braak, Hermen A. Toersche, André B. J. Kokkeler, and Gerard J. M. Smit. Adaptive resource allocation for streaming applications. In L. Carro and A. D. Pimentel, editors, *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2011, Samos, Greece*, pages 388–395, USA, July 2011. IEEE Circuits & Systems Society. doi: 10.1109/SAMOS.2011.6045489.

[TDtB:6] Tapani Ahonen, Timon D. ter Braak, Stephen T. Burgess, Richard Geißler, Paul M. Heysters, Heiki Hurskainen, Hans G. Kerkhoff, André B. J. Kokkeler, Jari Nurmi, Gerard K. Rauwerda, Gerard J. M. Smit, and Xiao Zhang. CRISP: Cutting Edge Reconfigurable ICs for Stream Processing. In J. M. P. Cardoso and M. Hübner, editors, *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, pages 211–238. Springer Verlag, London, 2011. doi: 10.1007/978-1-4614-0061-5_9.

[TDtB:7] Timon D. ter Braak. Using guided local search for adaptive resource reservation in large-scale embedded systems. In Gerhard Fettweis and Wolfgang Nebel, editors, *Proceedings of the Conference on Design, Automation and Test*

in Europe (DATE 2014), pages 158:1–158:4, 3001 Leuven, Belgium, March 2014. European Design and Automation Association. ISBN 978-3-9815370-2-4. doi: 10.7873/DATE.2014.171.

[TDtB:8] Timon D. ter Braak, Gerard J. M. Smit, and Philip K. F. Hölzenspies. Dynamic resource allocation. Technical Report TR-CTIT-16-08, Centre for Telematics and Information Technology, University of Twente, Enschede, February 2016.

[TDtB:9] Gadi Aleksandrowicz, Eli Arbel, Roderick Bloem, Timon ter Braak, Sergei Devadze, Görschwin Fey, Maksim Jenihhin, Artur Jutman, Hans G. Kerkhoff, Robert Könighofer, Jan Malburg, Shiri Moran, Jaan Raik, Gerard Rauwerda, Heinz Riener, Franz Röck, Konstantin Shibin, Kim Sunesen, Jinbo Wan, and Yong Zhao. Designing reliable cyber-physical systems. In *Proceedings of the Forum on specification & Design Languages (FDL) 2016*, September 2016.

RUN-TIME MAPPING

DYNAMIC RESOURCE ALLOCATION IN EMBEDDED SYSTEMS

High degrees of both logical and physical decentralization can easily have extremely complex dynamics which result in chaotic behavior; avoiding chaos while maintaining high performance and adaptivity in such systems with many degrees of freedom requires sophisticated control techniques.

– E. Douglas Jensen

Capacity planning is a set of functions concerned with determining and maintaining the balance between the workload and equipment configuration at a minimum cost consistent with throughput, response time and reliability objectives.

– Software Engineering Institute

Run-time mapping is a capacity planning methodology that is able to maintain adaptivity in large-scale embedded computing systems.

– Timon D. ter Braak

ISBN 978-90-365-4213-5