Application Task and Data Placement in Embedded Multi-core NUMA Architectures

Optimization techniques for the Samsung 16-SRP

Karl Viring
Abstract

Application Task and Data Placement in Embedded Multi-core NUMA Architectures

Karl Viring

The evolution of microprocessors has lead to a situation where more memory is integrated closer to the computational cores. This has created architectures where memory latencies vary depending on the calling cores location. Such architectures are referred to as Non-Uniform Memory Access (NUMA) architectures. This adds further complexity to the already complex environment of developing parallel applications.

In this paper I research effective task and data placement optimization techniques for a Samsung Multi-Processor System-on-Chip (MPSoC) prototype. The research was structured by first conducting a series of extreme case micro benchmarks to gain insight of hardware behavior. These insights was then used to optimize two applications from the imaging domain; a 2D image blurring application and a 3D Seeded Region Growing (SRG) application.

The results from conducted benchmarks show that a wide range of factors are of importance when optimizing applications for the Samsung 16-SRP architecture. Although NUMA penalties exists, reducing congestion at the memory controllers and in the DMA channels are of importance to overall execution time. I propose task and data distribution schemes that work well for benchmarks with static and dynamic workloads. Clustered hierarchical work queues with work stealing have shown to be an effective approach to optimizing applications with a dynamic workload.

For future research it would be interesting to run further micro benchmarks of the system under congestion. To gain further verification of suggested task and data distribution schemes suggested in this thesis it would be of interest to apply them to more applications.
Acknowledgements

First of all I want to thank my supervisor Prof. Bernhard Egger for taking his time and allowing me to come to the Computer Systems and Platforms laboratory to work on this thesis project. Prof. Egger’s support and commitment to this project has been invaluable.

I also want to thank Prof. Sang Lyul Min and Prof. Anders Berglund, and their respective institutions, for making this first time ever thesis exchange between Seoul National University and Uppsala University possible. In addition I also would like to thank coordinator Olle Eriksson and Prof. Wang Yi at Uppsala University for their roles in reviewing and examining my master thesis.

Last but not least I also want to thank Samsung Advanced Institute of Technology for letting us work with the Samsung 16-SRP prototype. I want to especially thank Sangheon Lee for his cooperation and helpful support.
# Contents

Abstract i  
Acknowledgements iii  
Contents iv  
List of Figures vi  
List of Tables viii  
Acronyms ix  
1 Reading instructions 1  
2 Introduction 2  
  2.1 Background 2  
  2.2 Project overview 3  
  2.3 Problem statement 4  
  2.4 Delimitations 4  
3 Samsung 16-SRP system overview 5  
4 Related studies 7  
  4.1 Data placement and task affinity 7  
  4.2 Global scheduling and task queue organization 9  
    4.2.1 Static and dynamic scheduling 9  
    4.2.2 Dynamic scheduling & run queue organization 10  
  4.3 NUMA-ratio 12  
5 System performance analysis 13  
  5.1 Memory latency benchmarks 13  
  5.2 Network-on-Chip & memory controller congestion benchmarks 15  
  5.3 DMA channel performance benchmarks 17
6  Optimizing Imaging Application Benchmarks  20
   6.1  Image blurring  ........................................ 20
   6.2  Seeded Region Growing (SRG) ............................ 23

7  Conclusion & future research  29

Bibliography  31

Appendix  33
List of Figures

3.1 FPGA prototype hardware (side) ........................................ 5
3.2 FPGA prototype hardware (top) ........................................ 5
3.3 Samsung 16-SRP architecture overview ................................. 6

4.1 Classification of scheduling methods .................................... 10

5.1 NUMA factors for accessing SDRAM from all cores in cluster 0 .. 14
5.2 NUMA factor symmetry for all cores accessing SDRAM ............... 14
5.3 Memory cost of accessing different memory locations relative to SPM. ............................................................................ 15
5.4 Network-on-Chip congestion benchmark design ............................ 16
5.5 Network-on-chip congestion performance relative to uncongested core accessing Nearest SDRAM ........................................... 16
5.6 Clustered Network-on-Chip congestion benchmark design ............. 17
5.7 Network-on-chip clustered congestion performance relative to uncongested cluster accessing Nearest SDRAM .......................... 18
5.8 DMA channel performance speedup relative to all cores fetching from the same SDRAM ......................................................... 19

6.1 Input image ............................................................................ 21
6.2 Blurred image result ............................................................... 21
6.3 Non-optimal image blurring performance for three data placement techniques ........................................................................ 22
6.4 Optimal image blurring performance for three data placement techniques .......................................................... 23
6.5 SRG performance benchmark for all three images ....................... 26
6.6 SRG task distribution over clusters, for all input images and algorithms ........................................................................... 27
6.7 Maximum cluster deviation from perfect load balancing for all input images ......................................................... 28
List of Tables

6.1 Image blurring execution time (million cycles) . . . . . . . . . . 22
6.2 Seeded Region Growing execution time (million cycles) . . . . . 26
Acronyms

CGRA  Coarse-grained Reconfigurable Architecture.
CSAP  Computer Systems and Platforms Laboratory.
DMA   Direct Memory Access.
MPSoc Multi-Processor System-on-Chip.
NoC   Network-on-Chip.
NUMA  Non-Uniform Memory Access.
SPM   Scratchpad Memory.
SRG   Seeded Region Growing.
SRP   Samsung Reconfigurable Processor.
1 Reading instructions

Chapter 2 gives a brief introduction to the background of this master thesis project and formulate a problem statement. In chapter 3 an overview is given of the Samsung 16-SRP system used for conducting this research. Chapter 4 discusses previous studies in the field and how they contribute towards the work conducted for this master thesis project. In chapter 5 the approach used for finding optimization techniques for the studied system is given, by presenting a series of extreme case micro benchmarks and following results. In chapter 6 two real-world imaging applications are presented and how the findings from the extreme case micro benchmarks of the previous chapter was applied onto them. Finally in chapter 7 a conclusion is provided together with suggestions for future research.
2 Introduction

2.1 Background

The evolution of modern microprocessors for both computers and handheld devices has led to architectures with more and faster cores. This unlocks an increased computational power, but also puts additional restraints on memory. As the number of cores and the computational power in a microprocessor increases, the memory is required to serve the microprocessor at an increasing rate. According to Park et al. [11] the decrease rate of SDRAM memory access latencies can't keep up with the increase rate of computational power. This memory bottleneck problem has been referred to as the memory wall [17]. So far a solution to this problem has been to integrate more high access speed memory on the chip. A consequence of closely integrated memory is that memory latency may vary depending on the location of the calling cores. Such systems are referred to as Non-Uniform Memory Access (NUMA) architectures.

Placing memory modules spatially nearby the accessing core reduces memory latencies, but further increases the complexity of many-core environments. Developing for many-core architectures with a network-on-chip and memory latencies that exhibit NUMA puts additional requirements on the runtime environment or the application programmer. Löl & Holmgren [5] have identified that the new challenges imposed by NUMA architectures need to be handled by either extending the computer system or the programming model. To avoid penalties from costly NUMA accesses, task and data distribution needs to be taken into consideration [1]. Achieving affinity between tasks and data is desirable to minimize the cost of accessing the memory and thus overall program execution time.

This paper researches an efficient programming model for a prototype embedded multi-core microprocessor for mobile devices, the Samsung 16-SRP. The Samsung 16-SRP was analyzed through a series of extreme case micro benchmarks to identify system specific characteristics. More specifically the goals of these benchmarks were to gain insight of the memory access latency depending on the location of the issuing core and the accessed memory, congestion on the memory controllers, and the Direct Memory Access (DMA) channels.

Findings from conducted micro benchmarks was then applied to two parallelized
real-world applications. One is a blurring application that blurs a 2D image; the second one is a 3D Seeded Region Growing (SRG) algorithm from the medical imaging domain. The blurring application calculates a new color value for each pixel based on the colors of neighboring pixels. Image blurring allows the work items to be statically divided among all cores, since the problem size is known prior to execution. The SRG algorithm, on the other hand, starts with a seeded pixel as the first work item. For each work item it checks whether the surrounding pixels meet the criteria for inclusion (typically based on color gradients). The pixels meeting the inclusion criteria are included in the region and added to the queue of work items. The region grows in three dimensions, adding each unvisited neighboring pixel along the x-, y- and z-axes. Since the pixels included in the region depend on the image data, it is not possible to compute a balanced workload statically. For both the image-blurring and SRG applications, parallelized 16-core implementations were used as a base case.

The results of the micro benchmark show that, in addition to the NUMA factor, congestion on the memory controllers and the DMA channels have a significant effect on overall system performance. For the blurring application, we have allocated the input data in four chunks of equal size to the memories behind the four memory controllers and then allocated the tasks to the cluster of cores that are located closest to their data. This greatly reduces congestion on the Network-on-Chip (NoC) and the DMA channels, and we observe a 3-fold speedup over the original application. For the SRG application, using a combination of private work queues and hierarchical shared local work queues with work stealing and an allocation scheme that puts new work items into the queue of the cores that are located closest to the source data, we achieve a 1.4- to 1.5-fold speedup for three input images depending on image distribution characteristics.

2.2 Project overview

This master thesis project was carried out at the Computer Systems and Platforms Laboratory (CSAP) at School of Computer Science and Engineering, Seoul National University, Korea. At the CSAP laboratory research is conducted towards future computer systems and platforms for computing devices. The CSAP laboratory has a history of conducting research projects in cooperation with leading industry partners.

For this project we had the opportunity of working with a MPSoC prototype provided by Samsung Electronics, constituting of 16 Samsung Reconfigurable Processor (SRP). The application area of the Samsung 16-SRP is to process computational intense tasks in embedded systems and handheld devices, such as tablet computers and smart displays. The Samsung 16-SRP hardware environment is further described in chapter 3.
2.3 Problem statement

This thesis report aims at finding effective ways of optimizing the programming model for parallel applications in embedded multi-core NUMA architectures. Methods for improving parallel applications are researched by identifying characteristics affecting memory performance through a series of extreme case micro benchmarks in a multi-core NUMA environment. These findings are then applied on two imaging applications to achieve optimizations through more effective memory usage, or more specifically the goal of this research is to achieve performance increases in means of reduced execution time through the following measures:

- Finding techniques for reducing memory access penalties through efficient data placement.
- Improving task distribution and load balancing through task scheduling techniques.
- Find methods to effectively use the network interconnect and memory controllers of the studied hardware environment.

2.4 Delimitations

This research focuses on achieving performance optimizations by minimizing penalties incurred from ineffective memory usage. More specifically by studying how usage of Samsung 16-SRP hardware resources together with task and data placement techniques affect the overall application execution time. Optimization techniques suggested in this paper will only be benchmarked for the Samsung 16-SRP, although that such techniques are of benefit to similar systems.

As the Samsung 16-SRP is provided as a prototype, currently no support for handling task and data placement is provided in by the computer system, such as in the compiler or the operating system. The build environment for the Samsung 16-SRP does not provide any message-passing interface such as OpenCL or OpenMP. The research of this project is thus limited at finding optimization techniques applied in the programming model. This does not mean that findings from this research can’t later be applied to other levels of the computer system.
3  Samsung 16-SRP system overview

For this work we had the opportunity to work with a Samsung Multi-Processor System-on-Chip (MPSoC) prototype. The FPGA prototype hardware is shown in Figure 3.1 and 3.2. The chip is aimed at next-generation high-performance handheld devices. The system consists of sixteen SRP [4] cores. A single SRP core is a combination of a VLIW processor and a Coarse-grained Reconfigurable Architecture (CGRA). It can seamlessly switch between the two modes depending on the requirements of the application. Control-flow intensive code segments with limited instruction-level parallelism run in VLIW mode. For data-intensive parts the compiler maps a modulo schedule onto the CGRA [9].

The sixteen SRP cores are organized in a mesh-grid structure. A NoC connects the cores to the four memory controllers. There is a dedicated network interface for each core. The four memory controllers are located in pairs of two on two opposing sides of the mesh. Each controller is connected to an SDRAM bank. Each memory controller occupies an area in the physical address space; the address of the memory request thus determines to which memory controller/SDRAM the request is sent by the NoC. Figure 3.3 displays the architecture of the Samsung 16-core SRP chip. The cores can be grouped logically into four clusters based on their nearest memory bank, where SRP0 to SRP3 belong to cluster 0, SRP4 to SRP7 belong to cluster 1, and so on.
The memory hierarchy of the Samsung 16-SRP constitutes of three levels:

1. **Global address space**: each memory controller maps a 512MB area into the global address space. The up to 2GB of SDRAM are non-cacheable and globally addressable.

2. **Local address space**: each core has access to a 62MB cacheable local address space.

3. **Scratchpad Memory (SPM)**: each core contains a 64kB local SPM.

A cache is only provided for the local address space, thus the system is comparable to a non-cache-coherent NUMA.

A chip-wide DMA controller with 8 channels provides direct memory copies from the global memory into the local SPMs. The DMA controller supports 2D and linked DMA transfers.

Per default the SRP compiler maps core-local variables into the cacheable local memory area of each core. Shared data and large data structures are allocated in the global memory. In the default settings, the data is mapped to the beginning of the global address space, i.e., the first 512MB are located in SDRAM0, the second 512MB in SDRAM1, and so on. The SPM only contains the contents of the stack on each core in the default configuration; the lower part of the SPM is available to and under the control of the application programmer.

The results of this work are applicable to other multi-core processors with a similar memory hierarchy. The architecture of the single cores (SRP cores in our case) is irrelevant for the outcome of this study.
4 Related studies

Multicore NUMA architectures and MPSoCs can vary significantly in terms of configuration complexity regarding both memory and CPU architectures. For example, there can be a heterogeneous or homogeneous composition of cores, the NUMA architecture can be either cache-coherent or non-cache-coherent, all cores can share a single or have multiple distributed scratchpad memories. As this is the first time anyone outside of Samsung Electronics has been able to develop hands on for the Samsung 16-SRP prototype, few studies have been done for a matching set-up. The provided build environment currently does not provide support for OpenCL, OpenMP or any other message-passing interface. Related work tends to emphasize on either compiler level optimizations or an abstraction of partitioning and scheduling activities to middleware or the OS layer. 

Researching efficient ways to optimize applications in the user level can still be of interest for future compiler or OS solutions for similar hardware environments. At the same time, contributions from previous research are of interest since they propose solutions to similar problems.

In this chapter related studies are presented and discussed under three subsections, Data placement and task affinity, Global scheduling and task queue organization, and NUMA-ratio. Data placement and task affinity introduces related studies discussing data placement and placement techniques applied to similar systems. Global scheduling and task queue organization introduces methods for scheduling tasks in multi- or many-core NUMA architectures. Finally NUMA-ratio introduces a performance measure of the penalty incurred from accessing remote NUMA memory locations.

4.1 Data placement and task affinity

Several studies have been made focusing on providing memory management frameworks for OpenMP such as Broquedis et al. [1] and Marongiu & Benini [6], which also take distributed SPMs into consideration. Marongiu & Benini [7] mentions configurations where SPM is used instead of caches, as in our case where cache-coherence is not supported. Marongiu & Benini argue that SPMs
have the advantage of higher predictability with constant access times and lower energy consumption.

Other studies focus on handling memory management in the operating system. Several operating systems such as Solaris, Linux and Windows, automatically allocate memory according to a first-touch policy, where the data is allocated closest to the core accessing it [14]. This might initially be desirable, but in the case where a task gets migrated to another core closer to another memory bank the memory costs increase. Terboven et al. [14] implemented a next-touch policy that moves data to where its going to be used next. Lof & Holmgren [5] similarly implemented a next-touch policy and achieved satisfying performance increases.

Marongiu et al. [8] have studied work distribution for 3D MPSoC architectures, in contrast to the 2-dimensional Samsung 16-SRP architecture. In comparison to 2D architectures, 3D stacking technologies imply lower memory latency and high memory bandwidth. For 2D architectures Marongiu et al. mention that frequent updates of SPM from SDRAM are necessary to minimize more expensive accesses directly to SDRAM. They use a two-step approach to achieve high data locality and a balanced workload. The first step involves an initial static compile-time scheduling based on memory reference. The second step involves data migration by allowing imbalance to be corrected by work queue stealing from neighboring cores. Marongiu et al. finds work stealing modestly beneficial, mainly due to an initial satisfactory work balancing. Several interesting observations can be made from their results. Achieving a good initial load balancing is possible for an image blurring application, but is a less optimal approach for the SRG application.

In the case of image blurring data can be partitioned evenly between cores by the application programmer as the workload is evenly distributed. For SRG the subset of total data that constitutes the computational region is unknown, thus it is infeasible to have an initial balanced data partitioning prior to application execution. As for the case with work stealing Marongiu et al. benefit from faster memory access due to the 3D stacking technologies compared to the Samsung 16-SRP.

Saidi et al. [12] studied optimization of data granularity in systems relying on SPM and DMA transfers. They focus on applications that work on 2D arrays of input and output data. The minimal granularity, or basic block, is defined as the smallest element for which the computational step can be carried out. To decrease the granularity for DMA transfers basic blocks are then grouped together in super blocks. Using double buffering, data fetching and the computational part can be overlapped to minimize transfer overhead. Depending on granularity and memory performance the system can either be characterized as computation regime, where the computational part is dominant over data transfers, or as transfer regime, where the transfer part is dominant. The time consumption of a DMA transfer can be split into the command initialization
phase and the data transfer phase. The command initialization phase is independent of the transfer data size. The data transfer phase is proportional to the data size, assuming no simultaneous transfer requests. Saidi et al. applies a model where latency increases with the amount of simultaneous transfers. In our benchmarks this behavior have only been observed for cases where several transfers access the same SDRAM. They argue that optimal superblock size increases with the number of processor, as the superblock size determine the duration of the computational part, which is used to overlap the increased duration of fetching the next superblock. Saidi et al.’s results are of interest to the type of problem in this study, but in contrast they have been run on a simulator and not on real hardware.

4.2 Global scheduling and task queue organization

In multi-core computer systems and parallel applications the partitioning and scheduling of the assignment tasks have significant impact on the achieved execution time. Uneven distribution of an applications parallel section leaves resources unutilized, as some cores are idle while other cores may carry a heavy burden [13].

4.2.1 Static and dynamic scheduling

There are different approaches to task scheduling depending on whether it is the job of assigning time slices in a single core system, or as in our case, assigning tasks between several cores in a multi-core system. Scheduling for single-core systems is referenced to as local scheduling, whilst scheduling by assigning tasks in a multi-core system is referenced to as global scheduling [13].

Global scheduling algorithms can be either static or dynamic as illustrated in Figure 4.1. Static scheduling is when task assignment is carried out before execution, often at compile time. In dynamic, a scheduling workload is instead redistributed among the cores during execution to achieve a more even load balance between the work items. Generation of optimal static scheduling can be an NP-complete problem. Then on the other hand, dynamic scheduling is associated with overhead costs due to inter-core communication [13].

Crosbie et al. [2] have studied compile-time static scheduling for embedded systems. Crosbie et al. focuses on increasing data locality for cache based systems, which differs from our system, as the Samsung 16-SRP NUMA architecture provides no cache-coherency. The suggested method applies loop and memory transformations to increase data locality.
The two applications for which optimization techniques are studied in this paper, image blurring and 3D SRG, differ in terms of knowledge available prior to execution. For image blurring the workload is known prior to execution, but for SRG the workload is unknown as the searched region boundary is unknown. Due to these differences in nature of the two algorithms we can make use of an approach similar to static scheduling for image blurring, and a dynamic scheduling approach for the SRG algorithm.

### 4.2.2 Dynamic scheduling & run queue organization

Dandamudi & Cheng [3] have studied the performance impact of run queue organization in multi-core NUMA systems. The run queue contains tasks waiting to be executed. The denomination of tasks can be used interchangeably with process or thread. Dandamudi & Cheng [3] have identified three types of run queue organizations: centralized, distributed and hierarchical:

- **Centralized run queue**
  With centralized run queue organization the run queue is kept in global memory accessible by all cores. The advantage with a centralized run queue results in near optimal load balancing, but the queue quickly becomes a bottleneck as the number of cores increase.

- **Distributed run queue**
  With a distributed run queue organization a run queue is allocated for each core, and thus the problem of queue access contention has been resolved. The drawback of a distributed run queue is the overhead costs of initial task placement. Dandamudi & Cheng [3] states that distributed run queue organization is preferred for fine granularity tasks.

- **Hierarchical run queue**
  Hierarchical run queues take advantage of the hierarchical properties in NUMA architectures. In a hierarchical organization run queues are organized in a tree structure on the different levels of memory available to the
core. For large granularity tasks and when frequent synchronization is not needed, hierarchical run queues have shown to outperform distributed run queues. Hierarchical scheduling can be effective to reduce inter-memory communication [16].

Several studies have been conducted implementing hierarchical run queues for NUMA architectures. Common for previous studies by Wang et al. [15] [16], Zhou & Brecht [18] and Olivier et al. [10] is that they have grouped cores together in the hierarchy depending on their affinity in a NUMA context. This allows nearby cores to take advantage of NUMA characteristics and to reduce inter-memory communication. Further introducing queues on several hierarchical levels decreases the amount of cores simultaneously placing locks on queues.

In a study by Wang et al. [16] two hierarchical migration policies based on data affinity are suggested. Initially all iterations are divided in chunks among clusters of nearby cores. When load imbalance occurs, migration is initially carried out within the cluster by stealing work from the most loaded core. Only when all cores in the local cluster have empty queues, tasks are migrated by stealing from other clusters.

Similarly Olivier et al. [10] suggests a hierarchical approach where data migration is conducted by a shepherd thread stealing for all cores sharing the same cache. Worker threads running on the same chip share a LIFO queue to maximize data locality and to decrease cache misses. Further Olivier et al. suggests lock-free queues such as double ended queues, deques, or partially ordered heaps for further reducing bottlenecks in the future.

Another study by Wang et al. [15] have studied scheduling and migration policies for cache-coherent NUMA systems focusing on increasing temporal and spatial task affinity. Wang et al. suggests a Clustered (CAFS) algorithm building on an earlier Affinity Scheduling Algorithm (AFS). AFS distributes chunks of iterations evenly to each core during its initial phase, and then allows cores to migrate iterations by stealing from the core with the heaviest burden.

To alleviate the cost of searching the heaviest work queue CAFS implements separate work queues for clusters of cores. In a later study by the same authors, Wang et al. [16] proposes HMAFS (Hierarchical Modified AFS) algorithm based on AFS and Modified AFS (MAFS), where migration is carried out hierarchically. HMAFS aims at further improve load balancing whilst taking NUMA penalties into consideration. In HMAFS the idle processor first searches the task queues of cores in its own cluster, before turning to more remote task queues. Our implementation of hierarchical clustered work queues with work stealing is similar to HMAFS, but aims at further improving locking behavior and reducing NUMA penalties. We prevent locking of work queues in the core level of the hierarchy by restricting SPM work queues from stealing. To still uphold balancing within the cluster we prioritize maintaining the cluster work
queue at a minimum length before queueing to the core-level work queue. Further, HMAFS chooses the most loaded cluster as a target for work stealing. We use an approach where inter-cluster stealing prioritizes work queues closer to the shepherd based on NUMA penalties. For example, cluster 0 would first try to steal from cluster 2, which has the lowest NUMA penalty for cores within cluster 0.

4.3 NUMA-ratio

In NUMA architectures where the access time to a local memory is smaller than the access time to a remote memory, the difference is illustrated by the NUMA-ratio [5]:

\[
NUMA\text{-ratio} = \frac{Remote\ access\ time}{Local\ access\ time}
\]

The NUMA-ratio thus gives a measure of the relative latency overhead incurred from accessing a more remote memory location.
5 System performance analysis

In order to find efficient ways for task and data placement it is essential to understand the underlying hardware and related memory access costs. Due to the prototype nature of the target system, little was known about its actual memory performance prior to this research. Characteristics such as memory bandwidth and latencies have significant impact on suggested data placement and redistribution strategies chosen. By conducting a series of extreme case micro benchmarks system-specific characteristics were evaluated.

5.1 Memory latency benchmarks

Penalties incurred from accessing a more costly memory location are of importance to data and task placement. An initial benchmark was conducted to measure memory latencies and corresponding NUMA factors for different memory locations, as well as to identify eventual memory interconnect heterogeneity. The NUMA-factor is calculated as the cost of accessing remote memory, divided by the cost of accessing the nearest memory. These benchmarks are needed to identify characteristics of the interconnect as well as for comparison between memory locations.

Latency benchmarks were carried out by issuing a series of load instructions of data stored in global memory, local memory and SPM. Access to global memory tests was carried out uncongested for each core by maintaining the rest of the cores in an idle state.

Latency benchmarks showed the existence of NUMA penalties in the range of 1.1 to 1.5. NUMA costs for cluster 0 (SRP 0 through 3 for which SDRAM 0 is local) accessing the four SDRAM locations are illustrated in Figure 5.1. The performance benchmarks showed highly similar results for all other clusters. For every cluster, there is a corresponding core with similar NUMA costs. The NUMA cost scale with the number of network interfaces on the path to the accessed memory. Network interfaces are available at each SRP core, as illustrated in Figure 5.1. This support that the memory interconnect is homogeneous. For optimization of imaging applications this suggests that remote memory accesses should be limited. Further it shows that specific remote memory locations are
preferred over others, which to a certain degree can be utilized for efficient data migration.

![Cluster 0 NUMA factors](image)

**Figure 5.1:** NUMA factors for accessing SDRAM from all cores in cluster 0.

Memory latency benchmarks showed that the Samsung 16-SRP memory interconnect is symmetric, which is further illustrated in Figure 5.2. Within each cluster there is a corresponding core with highly similar NUMA-factors.

![Network interconnect NUMA symmetry](image)

**Figure 5.2:** NUMA factor symmetry for all cores accessing SDRAM.

In addition benchmarks were performed accessing all levels of the memory hierarchy, including cacheable local memory. The effect of cache misses on local memory performance was studied by adjusting the spatial locality of the loaded elements to cause the desired frequency of cache misses.
The results show that SDRAM accesses incur a 4- to 5-fold higher cost than accessing data residing in SPM, illustrated in Figure 5.3. Further the SPM outperforms cacheable local memory, which in the worst case of 100% cache misses has a 5-fold higher cost. This supports that SPM should be preferred over cacheable memory as argued by Marongiu & Benini [8].

Figure 5.3: Memory cost of accessing different memory locations relative to SPM.

5.2 Network-on-Chip & memory controller congestion benchmarks

Memory intensive applications will cause frequent accesses to memory modules and the network interconnect. In order to gain insight about how congestion affects remote accesses, benchmarks of the Samsung 16-SRP in a congested state needed to be executed. The Network-on-chip benchmarks seek to identify congestion in the network interconnect, DMA channels and at the memory controllers.

The performance of the NoC was analyzed by fetching 10MB of data at 16kb per iteration, through DMA transfers between four cores and varying memory locations, with and without overlapping transfer paths. In addition, a benchmark where all four cores access the same SDRAM was carried out as a reference. Network interconnect benchmarks are illustrated in Figure 5.4.

The results from the different modes of access were compared to the latencies of an uncongested core accessing the same locations. For all configuration modes the benchmarks showed no access penalties compared to the uncongested state,
Figure 5.4: Network-on-Chip congestion benchmark design - (a) Nearest - no NoC overlap. (b) Vertical swap - NoC overlap. (c) Opposite SDRAM - NoC overlap.

as illustrated in Figure 5.5.

The reference benchmark where four cores access the same SDRAM instead implies that congestion is present in the memory controller. The benchmark shows that the memory access cost increases only by the NUMA-factor and that congestion at network nodes does not need to be taken into consideration when optimizing applications for the Samsung 16-SRP. Instead congestion at memory controllers needs to be considered.

Figure 5.5: Network-on-chip congestion performance relative to uncongested core accessing Nearest SDRAM.
In addition, similar tests were run with all 16 cores, adapting the same access pattern within each cluster, as illustrated in Figure 5.6. This case is similar to previous tests with one calling core per cluster, but the potential of congestion at memory controllers and network interconnect bottlenecks is increased 4-fold as the number of participating cores increase.

![Figure 5.6: Clustered Network-on-Chip congestion benchmark design - (a) Nearest - no NoC overlap. (b) Vertical swap - NoC overlap. (c) Opposite SDRAM - NoC overlap.](image)

These clustered tests further confirm that simultaneous traffic in the network interconnect does not affect memory performance, as the uncongested cluster shows equal performance, as illustrated in Figure 5.7. The benchmark puts the memory controllers under additional congestion, as four cores concurrently issue memory load operations to the same memory controller. A consequence of this is that differences in NUMA-factors can no longer be observed, as the rate of which the memory controller can serve the channel is the limiting factor. NUMA-factors still need to be considered when optimizing applications for the Samsung 16-SRP, as congestion naturally will be less for applications containing a computational part.

### 5.3 DMA channel performance benchmarks

To test the impact of DMA channel assignment when all cores access memory, two modes of DMA channel assignment was tested; (1) assigning DMA channels
to pairs of cores in order of the core id, grouping core 0 and 1 together etc., and (2) assigning channels in an enumerated fashion, based on their core id modulo 8, grouping core 0 and 8 together etc. Channel assignment was tested by running three different memory access modes, (1) letting all cores access the same SDRAM module, (2) letting all cores access all SDRAMS iteratively, and (3) letting all cores access the nearest SDRAM module of the cluster.

The benchmark results show a 4-fold speedup for clustered memory access when assigning DMA channels in the paired mode as illustrated in Figure 5.8. The enumerated DMA channel assignment shows a similar speedup in the iterative memory access mode.

These two modes have in common that the cores sharing a channel are accessing the same SDRAM, which implies that DMA channel congestion occurs when the channel is used to access more than one memory bank. The results show that assigning channels based on the memory node to be accessed is favorable, and that a clustered memory access pattern results in a significant speedup during heavy memory congestion.
Figure 5.8: DMA channel performance speedup relative to all cores fetching from the same SDRAM.
6 Optimizing Imaging Application Benchmarks

The results from the system performance analysis were applied to two memory intensive real-world applications, 2D image blurring and a 3D Seeded Region Growing (SRG) application. Whilst both require frequent memory access to large sets of input data, they differ in the possibility of statically partitioning computational tasks prior to execution. In the base case both applications are parallelized for 16-cores. Benchmarks of application execution verify that findings from the system performance analysis of the Samsung 16-SRP can be used to effectively optimize parallel applications.

6.1 Image blurring

For the purpose of testing different data placement techniques we used a simple image blurring application. The application blurs a 2-dimensional image by re-calculating the color value of each pixel based on color values of neighboring pixels. For image blurring it is known prior to execution that all pixels of the image will be visited. Thus it is possible to distribute the computational tasks to cores prior to execution since the problem size is known. Two versions of the image blurring application were benchmarked, one optimal implementation and one non-optimal implementation respectively. The difference between the algorithms is that the non-optimal blurring loads each pixel 9 times (one as computational, and eight times as neighbor of surrounding pixels), whilst the optimal image-blurring algorithm keeps the pixel values in registers and thus only needs to load each pixel 3 times. The algorithms also differ in the granularity of fetched data. The non-optimal algorithm transfers twice as much data as the optimal algorithm for each iterative step. Potential congestion of DMA channels was reduced by assigning DMA channels to cores based on which memory location the core will access to avoid that a channel gets congested by accesses to multiple memory locations. In addition DMA fetching was carried out using double buffering to overlap the memory transfer with the computational part of image blurring.

Due to a hardware problem in the current version of the Samsung 16-SRP DMA
transfers of a length greater than one byte are not supported from L1 SPM to
global memory. Due to the existence of this hardware fault the cost of writing
the blurred image back to memory instead had to be simulated during bench-
marks by issuing a DMA load from SDRAM to SPM of the same size. DMA
transfers are assumed to be of the same cost both when fetching and writing
from main memory. Image blurring was debugged and verified by running the
same algorithm but not using DMA when writing out the blurred results.

The two image blurring algorithms benchmarked assume blurring of a pixel
based on its eight neighboring pixels. For the optimal image blurring bench-
marks a simplified algorithm was used. This to reduce development complexity
but still maintaining algorithm complexity and memory operation costs. The
simplified optimal blurring algorithm recalculates the pixel value based on the
two neighboring pixels in each row. The algorithm still maintains the same
computational complexity and loads the same amount of data from SDRAM
to SPM. A subsection of the input image and the blurring results is shown in
Figure 6.1 and 6.2.

![Figure 6.1: Input image](image1)

![Figure 6.2: Blurred image result](image2)

The performance of the image blurring application was measured by allocating
the data in three different ways. In the baseline all data was allocated to a
single SDRAM. The baseline was then compared to (1) allocating the image
to all SDRAMs and let cores fetch data from different memory locations with no respect to NUMA penalties. The last data allocation and task distribution was to (2) partition the image statically into four parts, one for each SDRAM. Cores were then assigned tasks based on the affinity of the image data to take advantage of the lower memory access costs, by only accessing data allocated in the same cluster.

Table 6.1: Image blurring execution time (million cycles)

<table>
<thead>
<tr>
<th></th>
<th>One SDRAM</th>
<th>Clustered SDRAM</th>
<th>All SDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-optimal blurring</td>
<td>37.48</td>
<td>12.80</td>
<td>17.85</td>
</tr>
<tr>
<td>Optimal blurring</td>
<td>19.00</td>
<td>6.86</td>
<td>12.23</td>
</tr>
</tbody>
</table>

The performance benchmarks of the non-optimal image blurring application showed a near 3-fold speedup using a clustered task workload distribution and data partitioning as illustrated in Figure 6.3. Table 6.1 lists the absolute execution times for the benchmarks.

Figure 6.3: Non-optimal image blurring performance for three data placement techniques.

The optimal image blurring application on the other hand showed a 2.75-fold speedup for clustered task and data distribution relative to allocating the data in
one SDRAM, as illustrated in Figure 6.4. Although the optimal image-blurring algorithm is significantly faster in terms of absolute execution time, the speedup compared to one SDRAM is slightly less. The difference in speedup between the two algorithms can be explained by the relation between time consumed fetching data and the computational part. Double buffering allows overlapping memory transfers and the computational part. In a case where the computational part is faster than the memory transfer, the memory transfer would be a limiting factor. Previously conducted channel and data locality micro benchmarks showed that a transfer regime clustered algorithm theoretically could achieve a 4-fold speedup compared to one SDRAM. Speedups of near 3- and 2.75-fold can be explained by the image-blurring algorithms being slightly computation regime.

For the All SDRAM configuration, the two image blurring implementations showed a speedup of 2 and 1.5 respectively. This can be explained by decreased memory controller congestion and on average better data locality compared to the One SDRAM configuration.

![Image blurring (optimal) - Data allocation performance](image)

Figure 6.4: Optimal image blurring performance for three data placement techniques.

### 6.2 Seeded Region Growing (SRG)

The SRG application is often used in the medical imaging domain to detect the boundary of an organ from a 3-dimensional image. The algorithm starts with a seeded pixel as the first work item. Neighboring pixels meeting the criterion are included in the region, and new work items for these are created and added
to a work queue. Since the searched region is dependent on the input data, it is not possible to statically allocate a balanced workload prior to execution. Thus a dynamic workload distribution is needed.

Prior to this project an unoptimized SRG application had been developed for the Samsung 16-SRP. The application was developed in a simulator environment and not for real hardware. Due to inconsistencies between supported hardware operations in the simulated environment the application would not run on real hardware. As a consequence of this a simplified application was developed. The two applications differ in the complexity of calculating the inclusion criterion.

The application is implemented looping over two phases. During the first phase the cores wait for a new work item in the work queue(s) available to the core. The second phase is started when a work item is obtained. The core will then fetch image data into its SPM using a DMA transfer and process the work item. To decrease data fetching granularity, work items were implemented as superblocks, containing an initial superblock seed and surrounding pixels. New seeds within the superblock are immediately calculated, and seeds on the border of neighboring superblocks are added back to the work queue if not already visited.

DMA channels were assigned according to the source or destination SDRAM location, based on findings from DMA channel micro benchmarks. This to maintain that no channel was used for accessing multiple SDRAM locations simultaneously.

The application was implemented in four different modes of operation to benchmark the performance of the applied optimization techniques.

1. **One SDRAM with waterfall work item distribution**
   As the baseline for our benchmarks the entire image was allocated to one SDRAM bank. The first core starts processing the initial seed and passes newly created seeds to the following core in order of the core id. The last core passes its work items onto the first core in a bounded fashion. For example work items created by SRP0 will be queued to SRP1 and so on. All cores have separate work queues allocated in the same SDRAM bank as the image data.

2. **Four SDRAM with waterfall work item distribution**
   Similar to the baseline this algorithm distributes newly created work items to the following core in order of the core id. Image data is allocated to all four memory banks, and cores fetch data from the SDRAM nearest to them in terms of NUMA performance. All cores have separate work queues allocated in their closest SDRAM.

3. **Four SDRAM with clustered work item distribution**
In the third algorithm there are two levels of hierarchical work queues, at the cluster level and bounded work queues at the core level, respectively. Implementing several levels of work queues allows reducing the use of global locks, and at the same time take advantage of faster SPM access. The input data is partitioned in four even sized parts and allocated to separate SDRAMs. The cluster-level work queues are located in the nearest SDRAM to the cluster. New work items are queued to the cluster where the seeded image data is allocated to decrease NUMA penalties from fetching data from costly remote memory. If a cluster work queue meets a length criterion of minimum 10 items, and the core belongs to the same cluster, the work item may be queued directly to the local work queue of the core residing in SPM. Cores may only dequeue work items assigned to its own cluster, but enqueue work items to other clusters. No data migration is carried out in case of work item imbalance.

4. **Four SDRAM with clustered stealing work item distribution**

Similar to the previous algorithm with clustered hierarchical work queues, work items are enqueued in work queues shared within the cluster or in the queueing cores SPM. New work items are enqueued to work queues within the cluster where the image data is allocated. In case of work item distribution imbalance one core in each cluster acts as a shepherd, stealing work items from remote cluster level work queues on behalf of all cores within the cluster to reduce excessive memory congestion. If the remote work queue fulfills a stealing criterion, up to 20 work items are added back to the cluster work queue. Work items are not stolen if the remote queue is too short, and a maximum of half of the exceeding work items can be stolen. Pseudocode of the Four SDRAM with clustered stealing work item distribution algorithm is provided in Appendix A.

Benchmarks for the SRG application were obtained for three different input images. The three images were chosen to benchmark performance under different task distribution conditions.

- **Input image A** - an uneven shaped body.
- **Input image B** - an uneven shaped body shifted towards two quadrants of the image.
- **Input image C** - a centered symmetric cube.

The performance of the SRG algorithm is shown in absolute numbers in Table 6.2. Figure 6.5 illustrates the results normalized to the One SDRAM with waterfall work item distribution. Four SDRAM & waterfall distribution demonstrated a speedup around 1.2- to 1.3-fold over the baseline for all input images.
Table 6.2: Seeded Region Growing execution time (million cycles)

<table>
<thead>
<tr>
<th></th>
<th>Input image A</th>
<th>Input image B</th>
<th>Input image C</th>
</tr>
</thead>
<tbody>
<tr>
<td>One SDRAM, Waterfall</td>
<td>22.31</td>
<td>24.83</td>
<td>21.50</td>
</tr>
<tr>
<td>4 SDRAM, Waterfall</td>
<td>19.45</td>
<td>18.67</td>
<td>18.25</td>
</tr>
<tr>
<td>4 SDRAM, Clustered</td>
<td>18.98</td>
<td>32.09</td>
<td>17.26</td>
</tr>
<tr>
<td>4 SDRAM, Work stealing</td>
<td>14.78</td>
<td>17.54</td>
<td>15.07</td>
</tr>
</tbody>
</table>

Since task distribution and locking behavior is the same between the two waterfall algorithms, the speedup can be explained by the decreased memory controller congestion and the improved data affinity.

The clustered hierarchical algorithm without stealing shows a speedup close to 1.2-fold for input image A and C, but demonstrates a significant slowdown for input image B. After enabling work stealing the clustered hierarchical algorithm shows a significant improvement achieving a speedup between 1.4- to 1.5-fold. Work stealing recovers performance degradation due to load imbalance, and outperforms the four SDRAM waterfall algorithm for all input images. Notable is that although input image C is symmetric and centered and started with a centered seed, clustered algorithm performance still benefits from work stealing.

Figure 6.5: SRG performance benchmark for all three images.

Figure 6.6 illustrates the cluster work item distribution for each mode of operation for all input images. From studying the work item distribution for the clustered algorithm, which blindly assigns work items based on data locality, it
can be observed that the work items for image B are heavily unevenly distributed for the clustered partitioning. For Input image A the work items distribute more evenly across the four clusters. From Figure 6.6 it can further be interpreted that task distribution for the clustered no stealing implementation and image C is not perfectly balanced. This can be explained by the superblock granularity, as superblocks spanning over two or more quadrants are scheduled in the same cluster.

![SRG - Task distribution](image)

Figure 6.6: SRG task distribution over clusters, for all input images and algorithms.

For all input images it was possible to reduce the work item imbalance by task migration through work stealing. Figure 6.7 demonstrates the maximum cluster deviation from a perfect load balancing (total task items / n clusters). The deviation from perfect load balancing further shows that clustered stealing performs well at balancing the workload. The deviation from perfect load balancing for clustered work stealing is comparable to the two waterfall algorithms.

The results from the SRG application benchmarks show that an increase in speedup can be achieved through clustered data partitioning if the input image distribution and seed are balanced. It further shows that data imbalance can be corrected by implemented data migration through work stealing. Even though work stealing incurs an overhead cost, its shown that the performance gain from work stealing is greater than the overhead cost, as work stealing demonstrates improved load balancing and execution time for all tested input images.
Figure 6.7: Maximum cluster deviation from perfect load balancing for all input images.
7 Conclusion & future research

This thesis studies the effects of the location of data and congestion on memory access performance of a multi-core embedded processor. Through an extensive series of experiments performed with specifically crafted micro benchmarks we show that in a multi-core embedded chip the location of the issuing core and the accessed memory exhibits memory access costs similar to NUMA systems. In optimized applications, most memory accesses will go to the local SPM, with data being copied into the SPM and back to the global memory through DMA operations. We show that in such a scenario the selection of the DMA channel as well as the location of the accessed data is the main single limiting factor in order to achieve optimal performance.

We have applied this insight to two real-world applications from the medical imaging domain. For a 2D image blurring application, the work items can be statically partitioned and assigned to the cores that are located closest to the image data. Similarly, DMA channels are assigned based on the location of the source data. We achieve a 2.75- to 3-fold speedup on average, which is near the theoretical maximum (4-fold). For SRG, a clever allocation of work items to clustered hierarchical work queues with stealing yields a speedup of 1.4- to 1.5-fold for the tested input images.

The results show the importance of task allocation to cores and the necessity for a clever usage of global resource such as memory, on-chip networks, and DMA channels. Benchmarks of the four different SRG implementations have shown that a combination of both task and data placement techniques are available to optimize applications for embedded multi-core NUMA architectures. There is a wide range of bottlenecks, and optimization techniques accordingly, that can be considered when implementing for similar environments. These bottlenecks reside both in the memory usage domain, such as channel selection, NUMA penalties, memory controller congestion, memory hierarchies, as well as in the task distribution domain, such as locking behavior and work item redistribution.

Prior to this research there was a unoptimized SRG application available. This application had previously only been run in a simulated environment, and made use of operations not supported by the Samsung 16-SRP FPGA prototype. Because of this a new SRG algorithm with a less complex inclusion criterion was
developed. For future research it would be interesting to merge these two applications to further confirm the findings in this thesis under different computation complexities.

It would also be of interest to further optimize the SRG application using work stealing. One such optimization can be to further increase the number of hierarchies of the work queues. This would further reduce locking behavior, and possible reduce work stealing overhead, as the competition for shared resources would decrease.

The lack of hardware support for writing L1 SPM back to global memory constitutes a source of possible errors in conducted image blurring benchmarks. The blurred result written back to SDRAM consists of a quarter of all DMA memory transfers for non-optimal blurring, and half of the optimal blurring algorithm.

Currently, our optimizations are performed manually in the application. For future work, it would be of interest to automate scheduling and optimization techniques into a runtime environment such as OpenCL. A more generic scheduling implementation in a framework such as OpenCL will require that the scheduling environment, with queue lengths and boundaries, is dynamically configured.

Before implementing a runtime environment it would be wise to conduct more benchmarks and running additional applications to gain verification of the findings of this study, and to better understand the underlying hardware behavior. One such area where behavior is not fully known is to which degree NUMA-factors occur during memory controller congestion. To simulate a situation which would occur in a real-world application the complexity of these tests would increase, as real-world applications involve a computational part. Still, the benefits of a clustered approach have been demonstrated for the two imaging applications of this study.

Another area of interest for future research is to study how the granularity of DMA fetching affects performance. Varying the granularity might alter the relationship between the transfer and computational part of studied applications, as suggested by studies of Saidi et al. [12]. Less computational regime applications should benefit even further from suggested optimization techniques of this study.
Bibliography


Appendix A

Data: A 3D image containing an unmapped region and an initial seed.
Result: A mapping of a region in the image containing the initial seed.
Initialization:
while true do
  if SPM work queue not empty then
    dequeue superblock;
  else if cluster work queue not empty then
    lock cluster work queue;
    dequeue superblock;
    unlock cluster work queue;
  else if core is shepherd then
    select steal target;
    if target length > protected then
      lock target work queue;
      steal loot number of suberblocks;
      unlock target work queue;
      enqueue loot-1 superblocks;
      activate last superblock;
    end
  end
  if active superblock then
    DMA load image data;
    while active basicblock do
      if basicblock meets criterion then
        include basicblock in region;
        for each basicblock neighbor do
          if basicblock in superblock and unvisited then
            enqueue basicblock;
          else if basicblock not in superblock \&\& superblock is unvisited then
            select cluster target;
            lock target work queue;
            enqueue superblock;
            unlock target work queue;
          end
        end
        end
      end
      if entire region mapped then
        terminate;
      end
    end
  end
end

Algorithm 1: SRG with hierarchical work queues and work item stealing