Application Task and Data Placement in Many-core NUMA Architectures

Karl Viring[1], Sangheon Lee[2], Yeongon Cho[2], Soojung Ryu[2], Bernhard Egger[1]

[1]School of Computer Science and Engineering,
Seoul National University, Korea
{karl, bernhard}
@csap.snu.ac.kr

[2]Samsung Advanced Institute of Technology,
Samsung Electronics, Korea
{sheon0.lee, yeongon.cho, soojung.ryu}
@samsung.com

ABSTRACT

The advent of many-core chips in the embedded world imposes new challenges to programmers. One of the most important challenges to achieve optimal performance is that the variance in memory access time depending on the issuing core and the location of the data has to be taken into consideration in the already complex environment of parallel applications. In this paper we seek efficient ways of implementing dynamic task and data placement for embedded many-core systems that exhibit NUMA. Achieving a high affinity between a task and its data is desirable to minimize memory access cost and program execution time. This research was carried out using a many-core MPSoC FPGA prototype from Samsung. The MPSoC comprises 16 reconfigurable processors, global SDRAM and distributed scratchpad memory. Since no message passing interface or API's such as OpenMP is available for the Samsung 16-SRP, this paper focuses on optimizations in the application layer. To gain insight into memory performance, a series of micro benchmarks has been carried out. These benchmarks aim at discovering NUMA behavior, memory hierarchy latencies, architecture symmetry, DMA-transfer overhead and congestion. Accessing the SDRAM exhibits NUMA factors up to 1.45, however, the most important factor for achieving high performance is to minimize congestion in the NoC and at the memory controllers. We have applied these findings to two real-world 3D imaging applications, blurring and Seeded Region Growing (SRG). The base cases for both applications are parallelized for 16 cores. For image burring, we achieve a 2.75- to 3-fold speedup by distributing the image data to the four SDRAMs and statically assigning the work items to the cores closest to the data. Depending on input image, the SRG application shows a 1.4- to 1.5-fold speedup compared to the base case by applying dynamic work load balancing using hierarchical queues and work-stealing. The results emphasize the importance of considering the underlying structure of the memory architecture in order to achieve high performance on embedded many-core chips. The goal of our research is to in the future implement automatic workload distribution for a parallelization framework such as OpenCL.

ODES'13, February 23 - 24 2013, Shenzhen, China

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming – *Parallel programming.*
D.4.2 [**Operating Systems**]: Storage Management – *Storage hierarchies.*
D.4.4 [**Operating Systems**]: Communications Management – *Buffering.*

General Terms

Algorithms, Measurement, Performance, Experimentation

Keywords

NUMA, SPM, Many-core, MPSoC, embedded, Reconfigurable processors, Task placement, Data placement, Scheduling, Load balancing, Data migration, Image blurring, Seeded Region Growing

1. INTRODUCTION

The evolution of modern microprocessors for both computers and handheld devices has led to architectures with more and faster cores. 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. Memory latencies have not been able to keep up with the evolution of microprocessors. This memory bottleneck problem has been referred to as the memory wall [11]. 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.

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. 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. In this paper, we research an efficient programming model for a prototype embedded many-core microprocessor for mobile devices. Our research was conducted by performing a series of micro benchmarks to gain insight into the memory access latency depending on the location of the issuing core and the accessed memory, congestion on the memory controllers, and the DMA channels.

We have applied the knowledge obtained by running the micro benchmarks to two real-world applications from the medical

imaging domain. One is a blurring application that blurs a 3D image; the second one is a 3D Seeded Region Growing (SRG) algorithm. The blurring application calculates a new color value for each pixel based on the colors of neighboring pixels. Image blurring allows the input data to be statically divided among all cores, since the problem size is known prior to execution. The Seeded Region Growing (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 3 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, we have used parallelized 16-core implementations as our 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 big 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 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 nearly 1.4- to 1.5-fold speedup for three input images depending on image distribution characteristics.

The contributions of this paper are as follows:

- we analyze an embedded many-core prototype in terms of memory access latencies depending on the location of the issuing core and the accessed data, the structure of the NoC, and congestion through a series of micro-benchmarks.
- we apply the insights gained by the analysis to two real-world benchmarks and propose task distribution schemes that work well for benchmarks with a static and a dynamic workload.
- the results are obtained by running all benchmarks on a real system, an FPGA prototype of a 16-core chip.
- we show that although NUMA penalties exists, reducing congestion at the memory controllers and in the DMA channels are of importance to overall execution time.

These findings can be used when implementing an automatic workload distribution scheme for OpenCL and the like.

The remainder of this paper is organized as follows. Section II discusses related work. In Section III, we give an overview of the FPGA prototype, its many-core and memory architecture. Section IV describes the micro benchmarks and the analysis of the many-core system. In Section V, we apply the findings to two real-world applications and present the results. Finally, we conclude the paper and discuss future work in Section VI.

2. RELATED WORK

Multicore NUMA architectures and MPSoCs (Multi Processor Systems-on-chip) 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 we are the first outside of Samsung Electronics 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.

Several studies have been made focusing on providing memory management frameworks for OpenMP such as Broquedis et al. [1] and Marongiu & Benini [3], which also takes distributed SPMs into consideration. Marongiu & Benini [4] mentions configurations where SPM is used instead of caches, as in our case where cache-coherence is not supported. SPMs have the advantage of higher predictability with constant access times and lower energy consumption.

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.

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 interests since they propose solutions to similar problems.

Wang et al. [8] 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 a 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. [9] 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 reminds of HMAFS, but aims at further improving locking behavior and reduce NUMA penalties. We prevent locking of work queues in the core level of the hierarchy by making SPM work queues restricted from stealing. To still uphold balancing within the cluster we prioritize maintaining the cluster work queue at a specified length before queueing to the core level work queue. Further HMAFS chooses the most loaded cluster as 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.

Marongiu et al. [5] 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 neecessary to

minimize more expensive accesses directly to DRAM. 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. [7] 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.

3. SYSTEM OVERVIEW & MEMORY ARCHITECTURE

For this work we had the opportunity to work with a Samsung Multi-Processor System-on-chip (MPSoC) prototype. The chip is aimed at next-generation high-performance handheld devices. The system consists of 16 Samsung Reconfigurable Processor (SRP) [10] 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 [6].

The sixteen SRP cores are organized in a mesh-grid structure. A network-on-chip (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 1 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 belongs to cluster 0, SRP4 to SRP7 belongs 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.

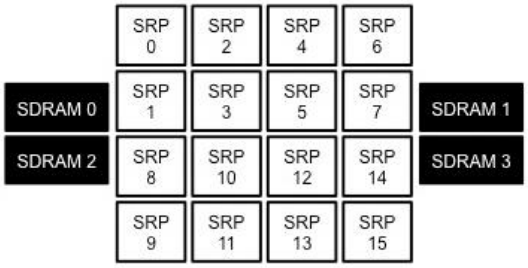

Figure 1. Samsung 16-SRP architecture overview.

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 many-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. 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 a significant impact on suggested data placement and redistribution strategies chosen. By conducting a series of extreme case micro benchmarks system-specific characteristics were evaluated.

4.1 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 were 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 accessing the four SDRAM locations are illustrated in Figure 2. 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 1. This supports 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.

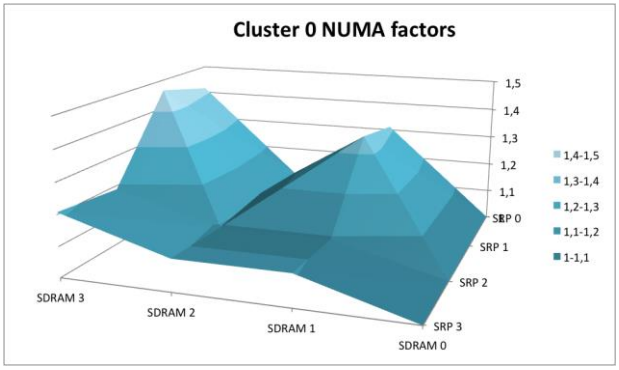

Figure 2. NUMA factors for accessing SDRAM from all cores in cluster 0.

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 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 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 [5].

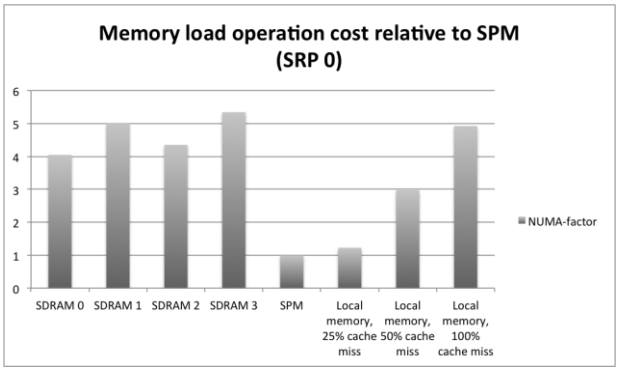

Figure 3. Memory cost of accessing different memory locations relative to SPM.

4.2 Network 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 4.

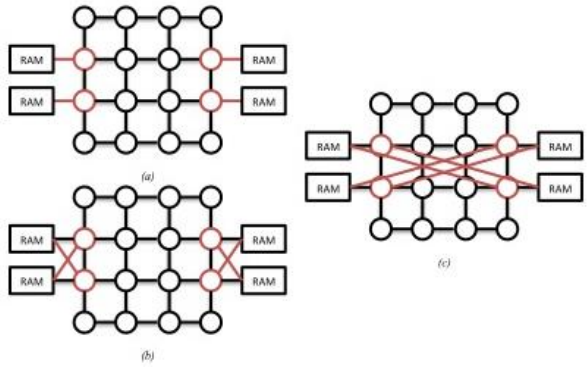

Figure 4. (a) Nearest - no NoC overlap. (b) Vertical swap - NoC overlap. (c) Opposite SDRAM - NoC overlap.

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, as illustrated in Figure 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 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.

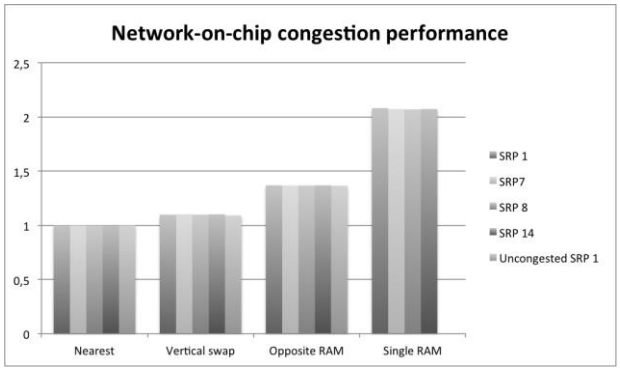

Figure 5. Network-on-chip congestion performance relative to uncongested core accessing Nearest SDRAM.

In addition, we ran similar tests with all 16 cores, adapting the same access pattern within each cluster. These clustered tests further confirmed that simultaneous traffic in the network interconnect does not affect memory performance, as the uncongested cluster showed equal performance.

4.3 DMA channel performance & memory congestion benchmarks

To test the impact of DMA channel assignment when all cores access memory, two modes of DMA channel assignment were 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 accessing the same SDRAM module, (2) letting all cores access all SDRAMS iteratively, (3) all cores accessing 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 6. 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.

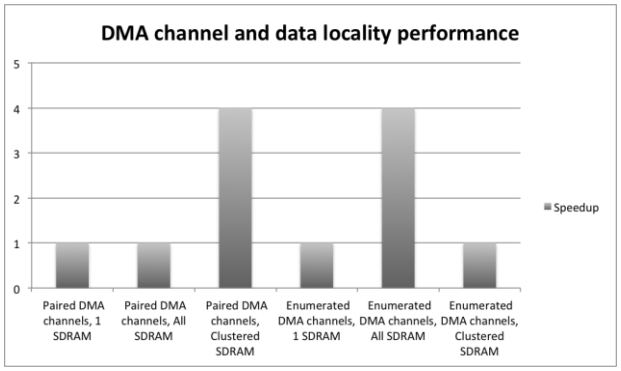

Figure 6. DMA channel performance speedup relative to all cores fetching form the same SDRAM.

5. OPTIMIZING IMAGING BENCHMARKS

The results from the system performance analysis were applied to two memory intensive real-world applications, 2D image blurring and a Seeded Region Growing 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 our 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.

5.1 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 suboptimal blurring visits each pixel 9 times (one as computational, and eight times as neighbor of surrounding pixels), whilst the optimal image-blurring algorithm visits each pixel 3 times in a sliding window fashion. Further the algorithms differ in granularity of the fetched data. The non-optimal algorithm transfers twice as much data as the optimal algorithm for each iterative step. Possible congestion of DMA channels were reduced by assigning DMA channels to cores based on which memory location the core will access, to avoid that a channel is congested by accessing 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.

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 in all DRAMs 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 1. Image blurring execution time (million cycles)

	One RAM	Clustered	All RAM
Non-optimal blurring	37.48	12.80	17.85
Optimal blurring	19.00	6.86	12.23

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 7.

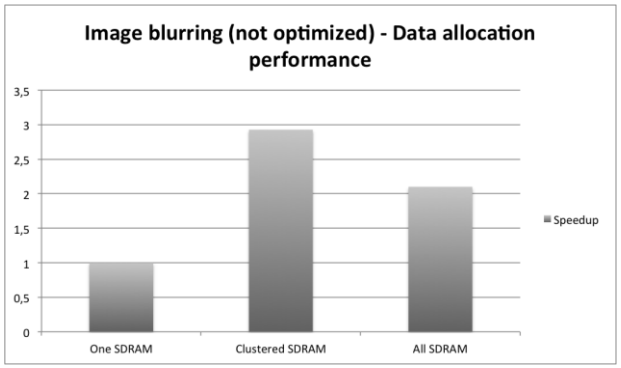

Figure 7. Non-optimal image blurring performance for three data placement techniques

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 dram, as illustrated in Figure 8. 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, image blurring showed a speedup of 2 and 1.5 respectively, which can be explained by decreased memory controller congestion and on average better data locality.

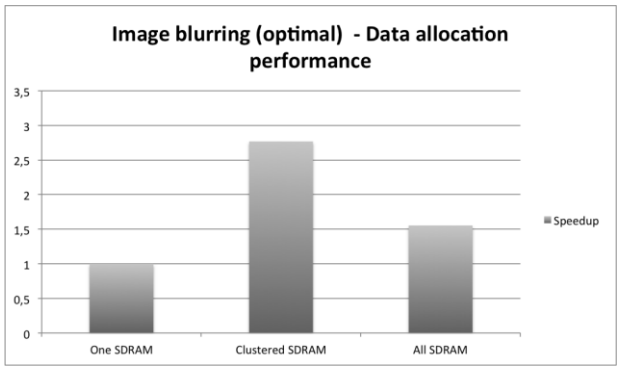

Figure 8. Optimal image blurring performance for three data placement techniques

5.2 Seeded region growing

The Seeded Region Growing (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.

The application was 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 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.

For DMA transfers channels were assigned according to the source SDRAM location, based on findings from DMA channel micro benchmarks.

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 hierarchical work queues**
   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 hierarchical work queues and NUMA optimized work stealing**
   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.

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. Image A – an uneven shaped body, Image B – an uneven shaped body shifted towards two quadrants of the image, and Image C – a centered symmetric cube.

Table 2. SRG execution time (million cycles)

	Image A	Image B	Image C
1 SDRAM, Waterfall	22.31	24.83	21.50
4 SDRAM, Waterfall	19.45	18.67	18.25
4 SDRAM, Clustered	18.98	32.09	17.26
4 SDRAM, Work stealing	14.78	17.54	15.07

The performance of the SRG algorithm is illustrated in Figure 9. All results are 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. 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.

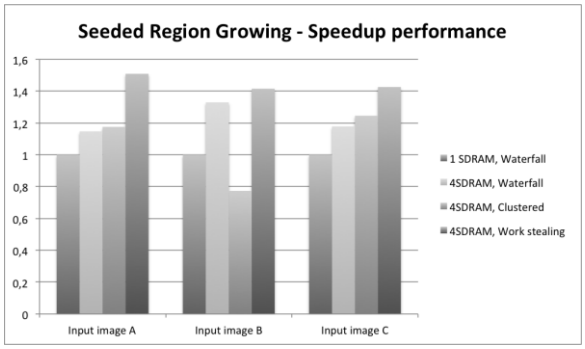

Figure 9. SRG performance benchmark for all three images.

Figure 10 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. Input image A's work items distribute more evenly across the four clusters. From Figure 10 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 granularity of the superblock, since superblocks spanning over two or more quadrants are scheduled in the same cluster.

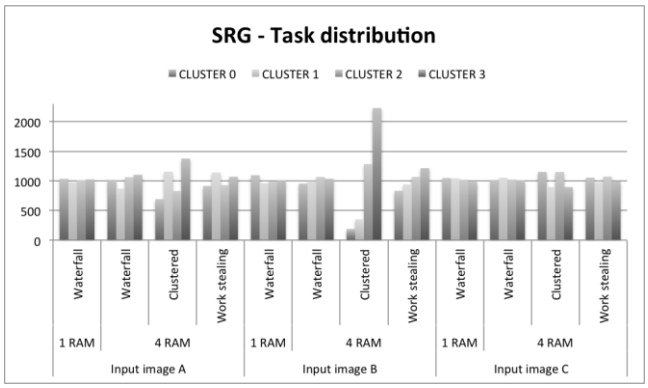

Figure 10. SRG task distribution over clusters, for all input images and algorithms.

For all input images it was possible to reduce the work item imbalance by task migration through work stealing. Figure 11 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 work load. The deviation from perfect load balancing for clustered work stealing is comparable to the two waterfall algorithms.

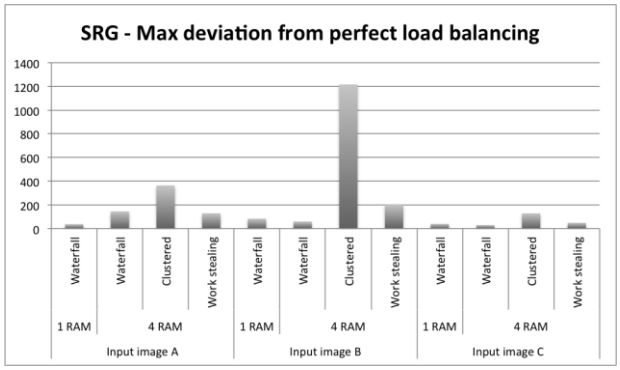

Figure 11. Maximum cluster deviation from perfect load balancing for all input images.

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.

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 many-core NUMA architectures. There is a wide range of bottlenecks, and tools 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.

6. CONCLUSION AND FUTURE WORK

In this work, we study the effects of the location of data and congestion on memory access cost for a many-core embedded processor. Through an extensive series of experiments performed with specifically crafted micro benchmarks we show that in a many-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.

These 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. Currently, our optimizations are performed manually and directly on the application level. For future work, we plan to automate this process and integrate it into our OpenCL runtime.

7. ACKNOWLEDGMENTS

We would like to thank the Samsung Advanced Institute of Technology for providing us with the opportunity to develop for the Samsung 16-SRP. ICT at Seoul National University provided research facilities for this study.

8. REFERENCES

[1] Broquedis, F., Furmento, N., Goglin, B., Namyst, R., & Wacrenier, P. A., Dynamic task and data placement over NUMA architectures: an OpenMP runtime perspective. Evolving OpenMP in an Age of Extreme Parallelism, 2009.

[2] Crosbie, N.E.; Kandemir, M.; Kolcu, I.; Ramanujam, J.; Choudhary, A.; , "Strategies for improving data locality in embedded applications," Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings. , vol., no., pp.631-636, 2002.

[3] Andrea Marongiu, and Luca Benini. "An OpenMP compiler for efficient use of distributed scratchpad memory in MPSoCs." Computers, IEEE Transactions on 61, no. 2, 2012.

[4] Andrea Marongiu, and Luca Benini. "Efficient OpenMP support and extensions for MPSoCs with explicitly managed memory hierarchy." In Proceedings of the conference on Design, automation and test in Europe, pp. 809-814. European Design and Automation Association, 2009.

[5] Andrea Marongiu, Paolo Burgio, and Luca Benini. Vertical stealing: robust, locality-aware do-all workload distribution for 3D MPSoCs. In Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems (CASES '10), 2010.

[6] Taewook Oh, Bernhard Egger, Hyunchul Park, and Scott Mahlke. Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures. In Proceedings of the ACM SIGPLAN/SIGBED 2009 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'09), Dublin, Ireland, 2009.

[7] Selma Saidi, Pranav Tendulkar, Thierry Lepley, and Oded Maler. "Optimal 2D Data Partitioning for DMA Transfers on MPSoCs." In Proceedings of the 15th EUROMICRO Conference on Digital System Design.

[8] Yi-Min Wang, Hsiao-Hsi Wang, Ruei-Chuan Chang, Clustered affinity scheduling on large-scale NUMA multiprocessors, Journal of Systems and Software, Volume 39, Issue 1, October 1997.

[9] Yi-Min Wang, Hsiao-Hsi Wang, Ruei-Chuan Chang, Hierarchical loop scheduling for clustered NUMA machines, Journal of Systems and Software, Volume 55, Issue 1, 5 November 2000.

[10] Won-Jong Lee, Shi-Hwa Lee, Jae-Ho Nah, Jin-Woo Kim, Youngsam Shin, Jaedon Lee, and Seok-Yoon Jung. SGRT: a scalable mobile GPU architecture based on ray tracing. In ACM SIGGRAPH 2012 Talks, SIGGRAPH'12, 2012.

[11] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. SIGARCH Comput. Archit. News, 23:20-24, March 1995.