Efficient Resource Management for Many-cores with Centralized L2 Caches using Distributed Control Processors

□ Authors : Surim Oh, Younghyun Cho, and Bernhard Egger

□ Institute : Department of Computer Science and Engineering, Seoul National University, Seoul, Korea

□ Contact Information : {surim, younghyun, bernhard}@csap.snu.ac.kr

1. Many-core Processor Architecture with Centralized L2 Cache

To handle many parallel applications, the massive computational power of GPUs or Coprocessor architecture such as Xeon Phi has been commonly used. While regular data parallel applications take advantages of such an architecture, the irregular parallel applications with various branch statements still suffer from poor parallelism and low data locality. To execute diverse parallel applications including irregular ones efficiently, many-core processor is possibly considered for the appropriate architecture.

There are several ways to configure many-core architecture for the sake of reducing communication overhead between many number of computing cores. As the size of data to be processed by application becomes bigger, the size of required on-chip memory of many-core processor is getting bigger. Non-Uniform Cache Architecture(NUCA) is commonly used for the huge shared cache. One thing to compose many-core architecture, therefore, is an efficient memory system configuration considering shared cache, which places L2 cache centralized and surrounded by computing cores to reduce the variation of cache access delays of NUCA.

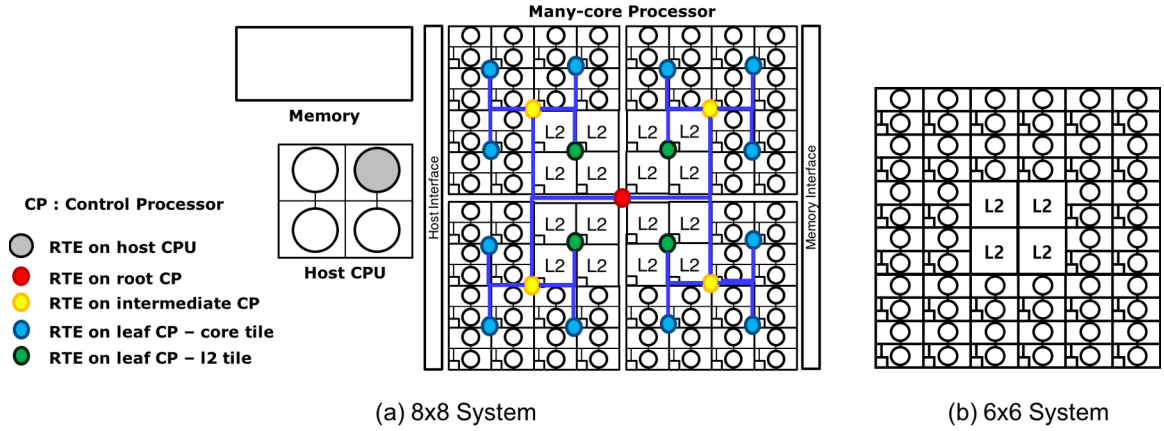

Figure 1. Simulated Many-core System

We introduce the simulated many-core architecture with centralized L2 cache and hierarchically placed control processors. The many-core processor which is the target of our proposed scheduler has centralized L2 cache placement. Figure 1 shows two

architectures of the simulated many-core system. One is a single many-core processor chip composed of 64 tiles, 48 core tiles and 16 shared L2 cache bank tiles. Each core tile has 2 cores, then a total of 96 cores exist in a single many-core processor chip. NoC(Network- on-Chip) design follows tree-mesh heterogeneous topology for low latency, where each network runs independently [1]. Control processors are hierarchically placed and used for hierarchical tile managements on our scheduling framework. The other is composed of 36 tiles, 32 core tiles and 4 shared L2 cache bank tiles. NoC design for this configuration follows mesh topology, and no control processor exists on the system.

2. Scheduling Framework

In this section, we introduce space-shared scheduler for simultaneously running multiple parallel applications on a many-core processor simulator. We present our scheduling software framework on the simulator.

A. System software stack

Our many-core RunTime Environment(RTE) is implemented on the many-core simulator. The runtime environment monitor running on host CPU communicates with application OpenCL API and sends the information of parallel applications which is required for scheduling. For example, RTE monitor sends each applications' execution information such as serial or parallel section to many-core architecture to execute proper OpenCL kernels when the application starts or ends, and when the parallel section of each application begins or ends. Then, the scheduler on the control processor performs space-shared core allocation for the parallel applications. The system software stack is shown in Figure 2.

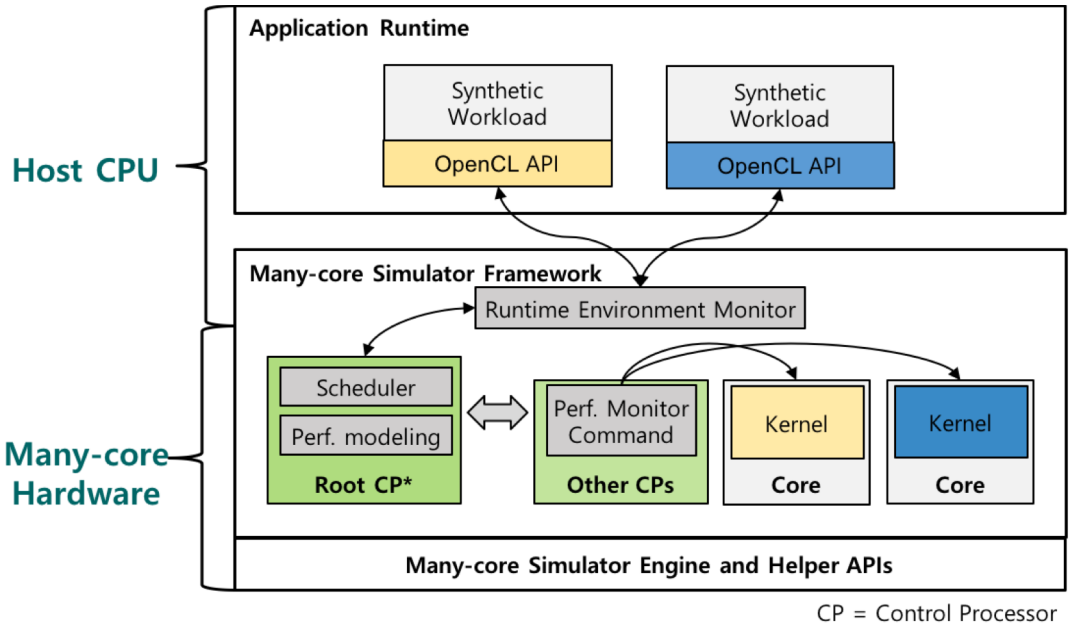

Figure 2. System Software Stack

We design two scheduling schemes for diverse parallel applications on many-core embedded systems, and validate the design with work-in-progress evaluation. One thing is that we distribute the works of a scheduler such as work-group scheduling to multiple chunks of works by using hierarchically placed control processors. The other thing is cache proximity-aware core mapping and core allocation based on the characteristics of simultaneously running applications. The goal of our proposed scheduler is not only scheduling a number of parallel applications with diverse characteristics but also achieving efficient load balancing.

B. Hierarchical Scheduling using Control Processors

Work-group scheduler for many-core systems should provide load balancing with low runtime overhead to maximize the performance of parallel applications. For example, load imbalance occurs if all the works are distributed statically when the applications begin. On the other hand, if the works are distributed dynamically one by one at runtime, the at-runtime scheduling overhead becomes big. Especially, as the number of cores becomes bigger, a fast work distribution by a single scheduler is getting harder. The hierarchical scheduling, therefore, is necessary for load balancing with low scheduling overhead at runtime. A root control processor placed on the

highest level among control processors performs coarse-grained work distribution which splits the works to the small number of big chunks for lower-level control processors. Intermediate control processors do fine-grained work distribution to leaf control processors, and the actual work-group distribution is done by leaf control processors. The root control processor and intermediate control processors achieve load balancing via work stealing on each level. Figure 3 shows our proposed distributed scheduling scheme.

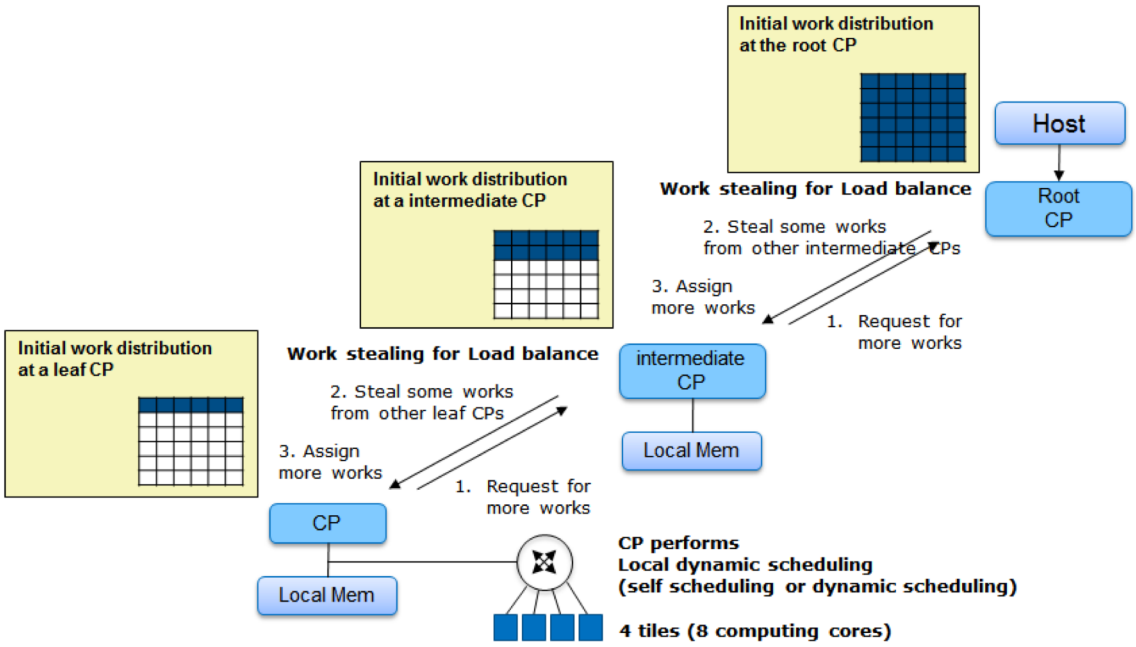

Figure 3. Scheduling Scheme using Control Processors

C. Core Allocation and Mapping Policies

Our proposed scheduler performs core allocation and mapping based on the characteristics of running applications. The characteristics of each application are obtained by monitoring hardware performance counters at runtime. We present two allocation policies using applications' characteristics, and the policies are as follows.

• Cache Proximity-aware Allocation - Dynamic equal core partitioning: per- form dynamic scheduling placing data-intensive kernels closer to L2 cache.

• Scalability-based Allocation - Dynamic core partitioning based on scalability and placing data-intensive kernels closer to L2 cache.

Based on the characteristics of running applications which can be obtained with negligible overhead by monitoring hardware performance counters, both the number of cores and the position of cores of each application can be decided.

3. Evaluation

A. Target Architecture

Architecture	8x8	6x6
Processor	Cortex-A5	Cortex-A5
Total cores	96	64
L2 cache size	8MB	2MB
# of L2 cache bank	16	4

Table 1. Target Architecture

Our scheduling schemes are evaluated on simulated 96-core processor and 64-core processor [2]. Our hierarchical scheduling is evaluated on 96-core with control processors, and the core allocation policies are evaluated on both processor configurations. The many-core simulator simulates the many-core processor which is described in Section 1.1. The main features of the architectures are shown in Table 1.

B. Synthetic Applications

Workload	Characteristics	Work Size	Job Description
A	Computation intensive	Small	Monte carlo PI estimation
B	Data intensive	Small	Load data to incur L2 accesses
C	Data intensive	Big	Load data to incur L2 accesses

Table 2. Target Application

For the evaluation, we generated two synthetic data parallel applications which have specific characteristics as Table 2. One has data-intensity, and the other has computation-intensity. The data-intensive workload loads data to incur L2 accesses, and the computation-intensive workload performs Monte Carlo PI estimation. To evaluate the performance for simultaneous applications, we measure the turnaround time of each application and the total turnaround time under each scheduling policy.

C. Scheduling Scenario

We evaluate distributed scheduling for three synthetic applications as shown in Table 2. For the validation of our core allocation scheme, two synthetic applications, data-intensive and computation-intensive each, are executed simultaneously.

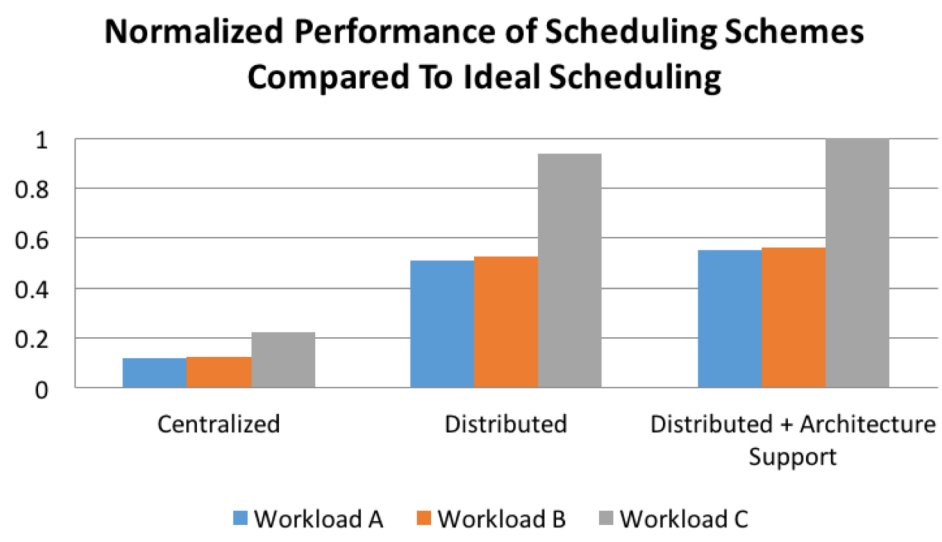

Figure 4. Synthetic Workload Result: 8x8 96-core platform.

Each bar presents workload A, B, and C respectively as shown in Table 2.

Figure 4 shows the benefits from distributed scheduling by using control processors. It shows the normalized performance of scheduling schemes compared to ideal scheduling, and the delays for each scheduling and communication method are

obtained by simulating the scheduling and communication as shown in Table 3. The result shows that there is a room for performance improvement of a parallel application which is data-intensive or computation-intensive.

Architecture	8x8	6x6
Processor	Cortex-A5	Cortex-A5
Total cores	96	64
L2 cache size	8MB	2MB
# of L2 cache bank	16	4

Table 3. Delay Annotation for a Schedule

Space-shared Core Allocation

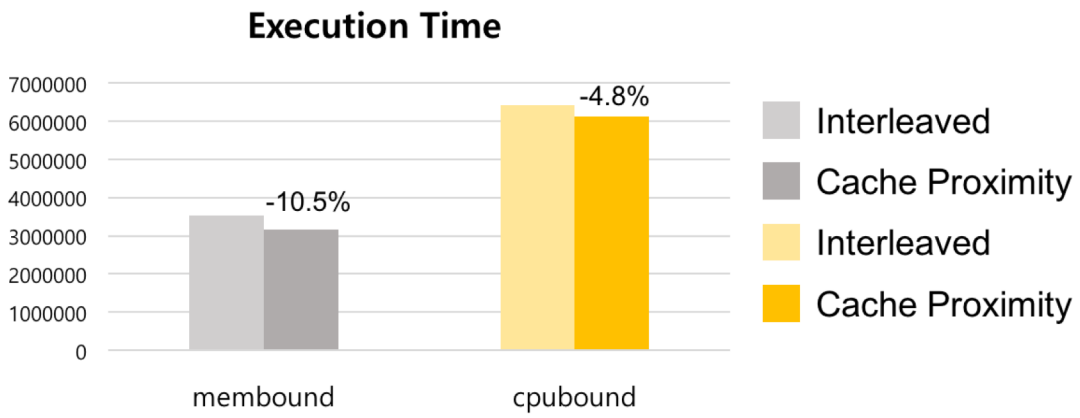

Figure 5. Cache Proximity-aware Scheduling Evaluation: 6x6 64-core platform. The unit of y-axis is absolute value of clock cycles. The grey bars present the data-intensive workload, and the yellow bars present the computation-intensive workload. The left bar of each color is for Interleaved Core Allocation, and the right one is for Cache Proximity-aware Allocation.

Figure 5 shows the performance improvement of cache proximity-aware core mapping policy with two parallel applications which are data-intensive and computation-intensive. Compared to Interleaved Core Allocation policy, Cache

Proximity-aware Allocation policy shows 10.5% and 4.8% performance improvement for data-intensive(membound) and computation-intensive(cpubound) applications respectively.

In Figure 6, different shapes of speedup curve appear between synthetic workloads. For data-intensive(membound) application, no more speedups from increasing cores can be obtained after the number of tiles becomes 16. Based on the scalability characteristics of each workload, our scheduler allocates different number of cores for each application.

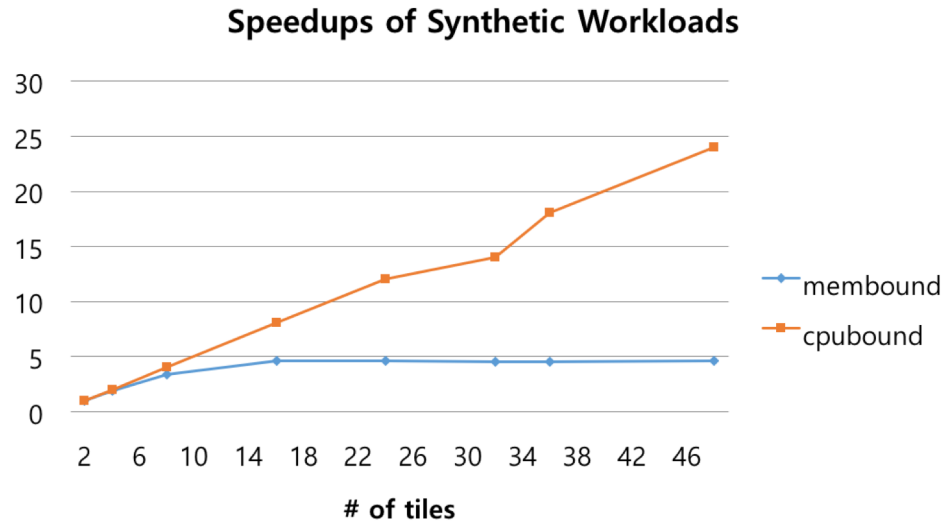

Figure 6. Speedups of Synthetic Workloads

Figure 7 shows the performance of scheduling policies compared to batch scheduling. 30.5%, 30.5%, and 38.5% improvements compared to batch scheduling are shown in the order Interleaved Core Allocation, Cache Proximity-aware Allocation, and Scalability-based Allocation. Although Scalability-based Allocation policy allocates the fewer number of cores to data-intensive workload, there are performance improvement for both workloads. The results imply that all the space-shared scheduling schemes are effective on many-core systems.

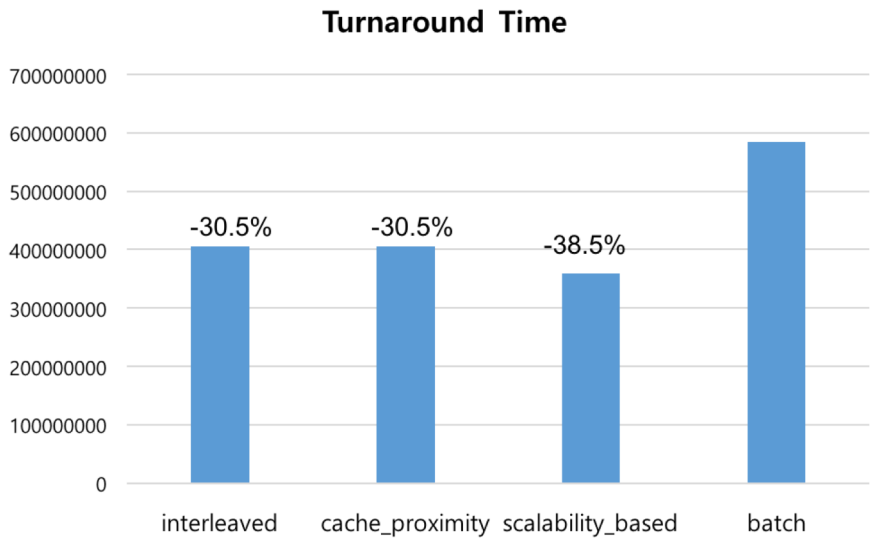

Figure 7. Scheduling Policies Result Compared to Batch Scheduling: 8x8 96-core platform. The unit of y-axis is absolute value of clock cycles. The results are in the order of Interleaved Core Allocation, Cache Proximity-aware Allocation, Scalability-based Allocation, and Batch Scheduling.

4. References

[1] Sungju Han, Jinho Lee, and Kiyoung Choi. Tree-mesh heterogeneous topology for low-latency noc. In Proceedings of the 2014 International Workshop on Network on Chip Architectures, pages 19–24. ACM, 2014.

[2] Shin-haeng Kang, Donghoon Yoo, and Soonhoi Ha. Tqsim: Afastcycle-approximate processor simulator based on qemu. Journal of Systems Architecture, 66:33–47, 2016.