Application-to-Core Mapping Policies to Reduce Memory Interference in Multi-Core Systems

Reetuparna Das* Rachata Ausavarungnirun† Onur Mutlu† Akhilesh Kumar‡ Mani Azimi‡
*University of Michigan †Carnegie Mellon University ‡Intel Labs
reetudas@umich.edu rachata, onur@cmu.edu akhilesh.kumar, mani.azimi@intel.com

Abstract

How applications running on a many-core system are mapped to cores largely determines the interference between these applications in critical shared resources. This paper proposes application-to-core mapping policies to improve system performance by reducing inter-application interference in the on-chip network and memory controllers. The major new ideas of our policies are to: 1) map network-latency-sensitive applications to separate parts of the network from network-bandwidth-intensive applications such that the former can make fast progress without heavy interference from the latter, 2) map those applications that benefit more from being closer to the memory controllers close to these resources. Our evaluations show that both ideas significantly improve system throughput, fairness and interconnect power efficiency.

Categories and Subject Descriptors: C.1.2[Computer Systems Organization] Multiprocessors; Interconnection architectures; C.1.4[Parallel Architectures] Distributed architectures
General Terms: Design, Algorithms, Performance
Keywords: Multicore, scheduling, interconnect, memory

1. INTRODUCTION

One important use of multi-core systems is to concurrently run many diverse applications. Managing critical shared resources, such as the on-chip network (NoC), among co-scheduled applications is a fundamental challenge. In a large many-core processor, which core is selected to execute an application could have a significant impact on system performance because it affects contention and interference among applications. Performance of an application critically depends on how its network packets *interfere* with other applications' packets in the interconnect and memory, and how far away it is executing from shared resources such as memory controllers.

While prior research (e.g., [9, 14, 13]) tackled the problem of how to map tasks/threads *within* an application onto cores, the interference behavior *between* applications in the NoC is less well understood. Current operating systems are unaware of the on-chip interconnect topology and application interference characteristics at any instant of time, and employ naive methods while mapping applications to cores. For instance, Linux 2.6.x [1] assigns a static numbering to cores and chooses the numerically-smallest core when allocating an idle core to an application. This leads to an application-to-core mapping that is oblivious to application characteristics and inter-application interference, causing two major problems.

First, overall performance degrades when applications that interfere significantly with each other get mapped to closeby cores. Second, an application may benefit significantly from being mapped to a core that is close to a shared resource (e.g., a memory controller), yet it can be mapped far away from that resource (while another application that does not benefit from being close to the resource is mapped closeby the resource), reducing system performance. To solve these two problems, in this work, we develop intelligent application-to-core mapping policies that are aware of application characteristics and on-chip interconnect topology.

PACT'12, September 19–23, 2012, Minneapolis, Minnesota, USA.
ACM 978-1-4503-1182-3/12/09.

2. OVERVIEW OF MAPPING POLICIES

Our policies are built upon two major observations. First, we observe some applications are more *sensitive* to interference than others: when interfered with, network-sensitive applications slow down more significantly than others [3]. Thus, system performance can be improved by separating (i.e., mapping far away) network-sensitive applications from aggressive applications that have high demand for network bandwidth.

Second, an application that is *both memory-intensive and network-sensitive* gains more performance from being close to a memory controller than one that does not have either of the properties (as the former needs fast, high-bandwidth memory access). Thus, system performance can be improved by mapping such applications to cores close to memory controllers.

We sketch our proposed mapping schemes with illustrations. Detailed descriptions of our policies can be found in a technical report [5]. Figure 1(a) shows the logical layout of a many-core processor interconnected with an 8x8 mesh NoC. Corner tiles consist of the memory controllers (triangles). Every other tile consists of a core, L1 cache and L2 cache bank. We divide the 8x8 mesh into four 4x4 clusters (marked by dotted lines).

1. Clustering In our policies, cores are clustered into subnetworks to reduce interference between applications mapped to different clusters. Applications mapped to a cluster predominantly access the memory controller within that cluster (and share the L2 cache slices within that cluster). To enable this, the system software's page mapping policies should be changed [5]. Clustering not only reduces interference between applications mapped to different clusters, but also 1) reduces overall congestion in the network, 2) reduces average distance packets traverse to get to the memory controllers or shared caches. Our results show most of the NoC packets of an application can be restricted to the application's cluster, with our page mapping algorithm, called Cluster-CLOCK [5].

2. Mapping Policy between Clusters An important question is which applications should be mapped to which clusters, as the answer affects which applications contend with each other. To illustrate the importance of this, Figure 1(b), (c), (d) show three possible application-to-core mappings. Each core tile in the figure is shaded according to the network intensity of the application: a darker tile corresponds to an application with a higher *L1 misses per thousand instructions (MPKI)*, i.e., a more network-intensive one. Figure 1(b) shows a possible random mapping of applications to clusters (called RND). This mapping is exemplary of policies usually employed in existing general-purpose systems. Unfortunately, it does not take into account 1) how applications mapped to the same cluster would interfere with each other, 2) balance of network and memory load across clusters, and as a result leads to high contention and relatively low system performance.

Figure 1(c) illustrates an example of a *balanced* mapping (called BL), which equally divides the network load among clusters. The idea is to distribute applications to clusters such that the sum of MPKIs in each cluster is similar across clusters. BL can achieve better performance than RND as it more effectively utilizes NoC and memory bandwidth. However, we find BL is not the best-performing mapping because it can map network-sensitive applications to the same cluster as network-intensive ones, leading to significant interference against, and thus slowdowns for, network-sensitive applications.

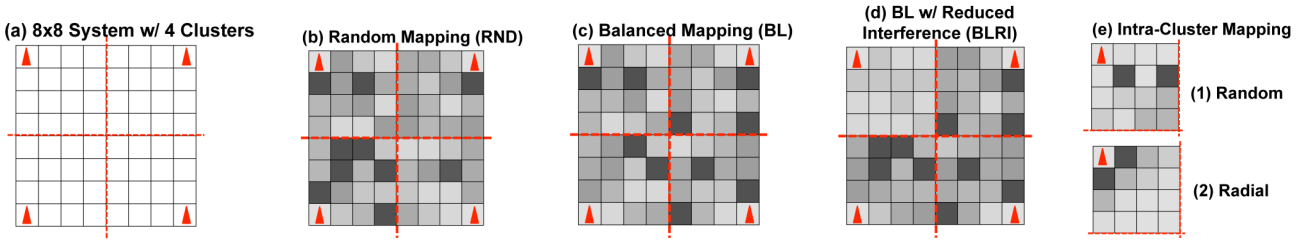

Figure 1: Visual examples of clustering and different inter-cluster and intra-cluster mapping policies

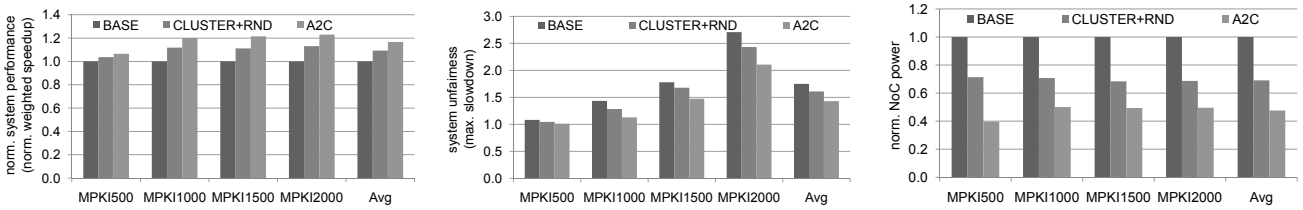

Figure 2: (a) System performance (b) system unfairness and (c) interconnect power of A2C for 128 workloads

Our proposal, *Balanced mapping with Reduced Interference (BLRI)* is shown in Figure 1(d). BLRI attempts to protect interference-sensitive applications from others by assigning them their own cluster (the top left cluster) while trying to preserve load balance in the rest of the network (i.e., among more network-intensive applications). BLRI can improve performance significantly because it can keep the slowdowns of the most latency-sensitive applications under control.

3. Mapping Policy within a Cluster After mapping applications to different clusters, a question remains: *which core within a cluster* should an application be mapped to? Figure 1(e) shows two possible intra-cluster mappings for a single cluster. *Random* mapping is not the best as it is agnostic to application characteristics. In contrast, our proposed mapping (*Radial*), places applications radially in concentric circles around the memory controller in decreasing order of a metric that considers both network-intensity and network sensitivity of an application. Darker (inner and closer) tiles represent network-intensive and interference-sensitive applications; lighter (outer and farther) tiles represent lower intensity applications with low sensitivity. We found the former type of applications benefit from being close to the memory controller, and this policy therefore improves performance.

Putting It All Together Our final policy, A2C, operates periodically by invoking the above three steps in order. It uses BLRI mapping across clusters and Radial mapping within a cluster. Our technical report [5] provides the policy details, examines tradeoffs, and describes implementation issues.

3. EVALUATION

We evaluate A2C using an instruction-trace-driven, cycle-level x86 CMP simulator [3, 4, 5] that models cores (with limited instruction window), interconnect, and memory. Our baseline configuration has 60 cores and 4 memory controllers connected with a 2D, 8x8 mesh NoC. We also use a detailed functional model for virtual memory management to study page access and page fault behavior of our workloads. The baseline page allocation and replacement policy is CLOCK [10]. We use our Cluster-CLOCK policy [5] to enforce clustering, which slightly reduces the page fault rate (an effect *not* included in performance results presented below).

We evaluate three systems: 1) the baseline with random mapping of applications to cores without clustering (BASE), 2) our enhanced system using clustering and Cluster-CLOCK, and random mapping of applications to cores (CLUSTER+RND), 3) our final system with A2C. We evaluate 128 multiprogrammed workloads of 35 diverse applications. Workloads are categorized into four groups based on their network intensity measured in terms of last-level cache MPKI.

Figures 2(a) and 2(b) respectively show system performance (higher is better) and system unfairness [11] (lower is better). Solely using clustering (CLUSTER+RND) improves weighted speedup by 9.3% over the baseline (BASE). A2C improves weighted speedup by 16.7% over the baseline, while reducing unfairness by 22%. Figure 2(c) shows the normalized average interconnect power consumption (lower is better). Clustering reduces power consumption by 31.2% over baseline; A2C by 52.3%. The clustering of traffic to memory controllers, reduced inter-application interference with A2C, and mapping network-intensive applications close to memory controllers altogether largely reduce the average hop count, and hence the energy spent in moving data over the interconnect.

4. CONCLUSION

We showed that mapping applications to cores in a manner that is aware of applications' characteristics significantly improves system performance, fairness, and energy-efficiency. Our policies are synergistic with interference reduction methods in NoC (e.g., [3, 7, 4, 16, 8]) and memory (e.g., [12, 2, 6, 15]), and we intend to examine this interaction in future work.

References

[1] Linux source code (version 2.6.39). http://www.kernel.org.

[2] M. Awasthi et al. Handling the problems and opportunities posed by multiple on-chip memory controllers. In *PACT-19*, 2010.

[3] R. Das et al. Application-Aware Prioritization Mechanisms for On-Chip Networks. In *MICRO-42*, 2009.

[4] R. Das et al. Aergia: Exploiting packet latency slack in on-chip networks. In *ISCA-37*, 2010.

[5] R. Das et al. Application-to-core mapping policies to reduce interference in on-chip networks. In *Carnegie Mellon SAFARI Technical Report No. 2011-001*, 2011.

[6] E. Ebrahimi et al. Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems. In *ASPLOS-XV*, 2010.

[7] B. Grot et al. Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-a-Chip. In *MICRO-42*, 2009.

[8] B. Grot et al. A QoS-Enabled On-Die Interconnect Fabric for Kilo-Node Chips. *IEEE Micro*, May/June 2012.

[9] J. Hu et al. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *ASPDAC*, 2003.

[10] S. Jiang et al. CLOCK-Pro: an effective improvement of the CLOCK replacement. In *USENIX*, 2005.

[11] Y. Kim et al. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA-16*, 2010.

[12] Y. Kim et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *MICRO-43*, 2010.

[13] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Euromicro*, 2003.

[14] S. Murali and G. D. Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *DATE*, 2004.

[15] S. Muralidhara et al. Reducing memory interference in multi-core systems via application-aware memory channel partitioning. In *MICRO-44*, 2011.

[16] G. Nychis et al. Next generation on-chip networks: What kind of congestion control do we need? In *HotNets-9*, 2010.