An Empirical Study on the NoC Architecture Based on Bidirectional Ring and Mesh Topologies

Jie Yin
Nagoya University
Nagoya, Japan
yin@ertl.jp

Ye Liu
Nagoya University
Nagoya, Japan
liuye@ertl.jp

Shinpei Kato
Nagoya University
Nagoya, Japan
shinpei@is.nagoya-u.ac.jp

Hiroshi Sasaki
Columbia University
New York, USA
sasaki@cs.columbia.edu

Hiroaki Takada
Nagoya University
Nagoya, Japan
hiro@ertl.jp

ABSTRACT

Along with the fast increase of the core numbers integrated on a single chip, reasonable design of interconnection which connects all the cores and other necessary components, is always critical for giving adequate play to many-core's processing potential. There are various topologies proposed as the candidates of inter-core connection, and generally employed topologies in many-core design are 2D mesh and bidirectional ring topologies. However, scalability bottleneck is appeared to be a vital problem which may disappoint users, because multithreaded applications cannot gain proportional speedup during execution along with employing more cores. In this research, performance evaluation, bottleneck locating and systematical comparison which are centered on the in-depth analysis of *scalability*, between these two widely used interconnection patterns using bidirectional ring based coprocessor Intel Xeon Phi 3120A and 2D mesh based processor TILE-Gx8036 are unfolded.

CCS Concepts

•**Computer systems organization** → **Architectures;** •**Networks** → Network types;

Keywords

Many-core; Bidirectional ring; 2D mesh; Scalability bottleneck;

1. INTRODUCTION

Many-core architecture is the state-of-the-art breakthrough for developing high speed processors. The topology design is directly correlated with whether the processing talent of certain interconnection topology based many-core processors and accelerators can be given full play. Moreover, parallelized applications are also confronted with many additional constraints for gaining ideal speedup from not only the intrinsic properties of hardware platforms, but also its design of the parallelization from the perspective of software. Therefore, *scalability*, whether multithreaded applications can achieve ideal speedup while using larger number of cores, is always an interesting topic to discuss.

For probing into the culprits of varied scalability of different NoC designs, we conducted a comparative study between bidirectional ring topology and 2D mesh topology by designing dedicated micro-benchmarks for network evaluation and analyzing selected representative benchmarks from PARSEC benchmark suite[1].

Our work contributes to the comparison of bidirectional ring NoC architecture and 2D mesh NoC architecture using highly sophisticated many-core products. The advantages and disadvantages of both architectures are clearly pointed out and visualized by benchmarking and analysis. We summarized and compared the scalability of both platforms using different thread-to-core mapping patterns. Also we have located the culprits of different scalability between two platforms, which can be used as a reference for the structure design of architecture-specific parallel applications.

The rest of this paper is organized as follows. Brief architecture introductions of experiment platforms are given in Section 2. Section 3 describes the result of evaluation and in-depth analysis of scalability on two platforms. The last section summarizes the conclusion of the current work and the conception of future studies.

2. EXPERIMENTAL PLATFORMS

2.1 Intel Xeon Phi 3120A Coprocessor

Intel Xeon Phi 3120 Coprocessor is based on Intel Many Integrated Core (MIC) architecture, which connects 57 in-order Intel Pentium cores running at 1.1 GHz, distributed memory controllers with two channels of each, distributed Tag Directories(TD) for managing global cache coherence and other I/O interfaces by three pairs of ring buses [2][3] as shown in Figure 1.

Xeon Phi 3120A Coprocessor provides four hardware thread

© 2016 ACM. ISBN X-XXXXX-XX-X/XX/XX.

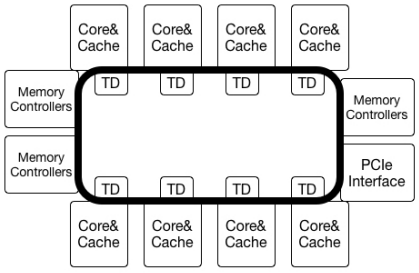


Figure 1: Logical Layout of Bidirectional Ring Architecture of Intel Xeon Phi 3120A Coprocessor.

contexts using hyperthreading technique, between which can be switched for executing in a round-robin manner [2]. All the threads can access 6 GB GDDR5 memory modules through interleaved requests to six memory controllers, which have 2 channel controllers of each and deliver message at 5.0 GT/s transfer speed. Therefore, the theoretical peak bandwidth of Intel Xeon Phi 3120A Coprocessor is 240 GB/s [4]. We measured the memory latency for each core of Xeon Phi using a simple pointer chasing micro-benchmark by setting process affinity using *sched_setaffinity()*. Xeon Phi owns streaming hardware prefetcher on the L2 cache controllers, but its functionality can be avoided by setting appropriate traversing stride. According to the measured results, the average latency of L1 data cache hit is around 2.8 ns, local L2 cache hit latency is around 22 ns and the average memory latency is around 275 ns. Moreover, the value of memory accessing latency measured on each core is correlated with each core's relative position. Those cores approaching the center show apparent lower memory latency than those cores which are closer to the edge as shown in Figure 2, in which the x-axis means the number of each core, and the y-axis means the corresponding memory accessing latency measured on each core.

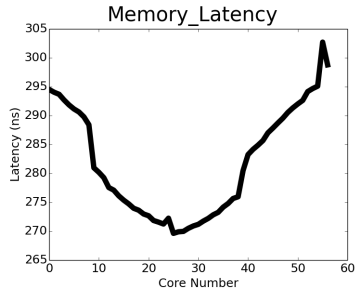


Figure 2: Respective Memory Accessing Latency of Cores 1 to 57 on Xeon Phi.

The network fabric of Intel Xeon Phi contains 3 pairs of wires. Each pair of wire consists of two identical wires traversing clockwise and counterclockwise. Different pairs handle different functions. Three pairs of wire are respectively named as BL, AD and AK[5]. BL wire is the most expensive one, which is mainly for transferring data blocks in 64 bytes wide. AD wire is smaller than data ring, which is for transferring command and address. AK has the smallest bandwidth, which is for transferring coherence messages and credits.

In order to have an in-depth analysis of how interconnection architecture can affect application's scalability, we designed micro-benchmark for Intel Xeon Phi to test the tolerance of bidirectional ring to the network congestion. This micro-benchmark employs a $57 \times 512K$ Bytes sized array which is aligned with the size of L2 cache subsystem, divided evenly into 57 chunks. Each chunk is assigned to a worker thread for caching the data chunk into its own L2 cache. *Mainworker*, defined as the first worker thread spawned by the main thread, is pinned to the core 1 and caches the first chunk into core 1's L2 cache. *Mainworker* spawns other *Subworker* threads, pinned to other cores respectively, for caching other chunks and writing to the *Mainworker*'s chunk simultaneously. Heavy cache coherence traffics are generated along with the increase of threads, which participate in the *Subworker* group. Result of benchmarking is shown in the Figure 3.

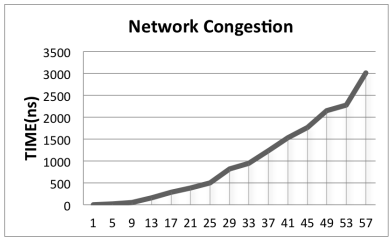


Figure 3: Result of Measuring Network Congestion on Intel Xeon Phi.

X-axis indicates the number of worker threads in total, and the y-axis shows the average time needed for an individual thread to finish its own operation. Along with more and more participants joining in the *Subworker* group, each thread has to consume much more time to finish its job due to the severe network congestion on the AK wire. Therefore, we conclude that bidirectional ring architecture is extremely sensitive to the network congestion, which is one of the intrinsic scalability bottlenecks.

2.2 TILE-Gx8036 Processor

TILE-Gx8036 processor owns 36 cores which work at 1.2 GHz, also named as tiles, organized by a 2D-mesh network 'iMesh'[6] into a 6×6 square. The logical layout of iMesh is shown in the Figure 4.

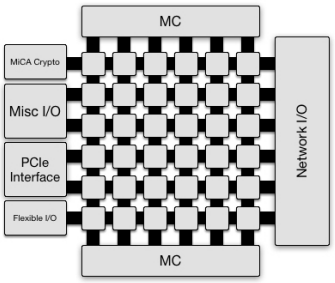


Figure 4: Logical Layout of iMesh of TILE-Gx8036 Processor.

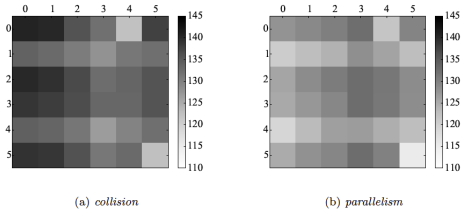
TILE-Gx8036 processor employs coherent L2 cache subsystems, and standard DDR3-1333 memory modules which can be accessed through two memory controllers using two

channels[7]. Users are provided the interface to choose whether to enable the use of memory striping for interleaving memory accesses to different memory controllers[7]. Theoretical peak memory bandwidth of TILE-Gx8036 is 21 GB/s. Memory read latency of TILE-Gx8036 is measured by Yixiao Li (mailto:liyixiao7@gmail.com) using a self-designed micro-benchmark. Main memory read latency with all the cache turned off is 103 ns(124 cycles) and read miss latency without cache coherence is around 98 ns (118 cycles). If the miss happened when requesting core is the home core of the requested cache line, accessing latency will become 105 ns (127 cycles). If the miss happened when requesting core is not the home core of requested cache line, accessing latency will reach 111 ns (133 cycles).

iMesh network of TILE-Gx8036 contains 5 types of wires for handling different works. Neighboring tiles are connected and intercommunicating using these 5 wires through the routing switches respectively integrated inside of each core.For explicitly visualizing the negative effects of network congestion on mesh network, another micro-benchmark is designed by Ye Liu, which is based on the concept that all the cores simultaneously read the large data set stored inside the main memory in the form of a linked list, with the whole cache system turned off. Each core is bound only one spawned thread by setting thread affinity. Only one page is used during execution in order to guarantee the least negative effect from TLB misses and the acquisition of accurate results. All the cores are vertically divided into left and right groups down the middle. There are two types of accessing patterns proposed in this experiment [8]:

- **Collision:** Each group accesses the memory controller at the opposite side.
- **Parallelism:** Each group accesses the memory controller at the same side.

The results after measuring are shown in the Figure 5. The color scales indicates the average time between sending read request and receiving the data.



(a) *collision* (b) *parallelism*

Figure 5: Network Congestion Visualizing on TILE-Gx8036 Processor.

It shows that cores spend much more time for accessing data when the accessed memory controller is at the opposite side(*Collision*). Under this scenario, data accessing requests merge on the network, which will cause the contention of router ports and memory controller accessing. However under *Parallelism* pattern, accessing time of each core is much relieved. We can observe that contentions on the network obviously slow down the memory accessing under *Collision* pattern.

Both architecture shows their negative reactions when network are busy at being occupied and mediating. This is a critical factor which may cause poor scalability on both architectures. However mesh network is more flexible to recover from resource competition because it has more input/output ports in the routers and multiple routing paths available for practical using. Bidirectional ring is vulnerable to the congestion because it only provides a pair of paths for all the streams.

3. SCALABILITY EVALUATION AND CULPRIT LOCATING

For further evaluating the scalability of both network architectures and locating the culprit of scalability bottleneck, 4 representative benchmarks, which have varied characteristics and parallel programming models, from PARSEC benchmark suite[1] are selected. These four benchmarks are *Blackscholes*, *Streamcluster*, *Dedup*, and *Canneal*. Based on the description in technical report [9] and research [10], we summarized the primary features of these four benchmarks in table 1. All the benchmarks are executed using the largest input set *native*, and *pthread* programming model. All the measurements implemented on Intel Xeon Phi excluded the using of the last core because it is where the μOS, the micro operating system for Intel Xeon Phi Coprocessor, is running.

We mainly designed four thread-to-core mapping patterns for each platform. For Intel Xeon Phi, the affinity patterns are implemented by setting pthread affinity attributes:

- **Serial.** Only one thread is pinned to each core. All the threads are pinned orderly and sequentially from core 1 to core 56.
- **OS.** No thread-to-core mapping is set for any thread.
- **Spread.** Only one thread is pinned to each core. The first worker thread starts from the midst core, and others spread to both sides step-by-step.
- **Double.** Two threads are pinned to each core. All the threads are pinned orderly from core 1 to core 28.

On the other hand, task-to-core mappings are implemented on TILE-Gx8036 by using *taskset* command line:

- **Center.** Tasks are initially mapped to the center tiles on iMesh and gradually spread to the edges and corners.
- **Spread.** Tasks are initially mapped to the four corner tiles on iMesh and gradually spread to the center.
- **OS.** Tasks are managed by OS without setting task affinity.
- **DDR0/DDR1.** Tasks are initially mapped to the tiles which are nearest to DDR0 or DDR1 and gradually spread to the opposite side.

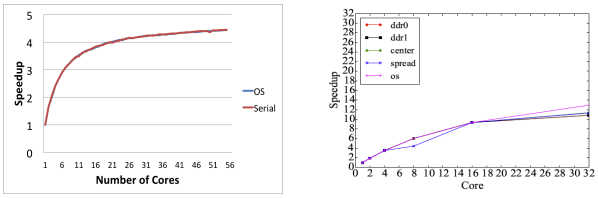
Based on these different mapping patterns, we analyzed the scalability of four selected benchmarks by doing the comparison not only between different mappings within each architecture, but also between these two architectures. All the experiment results of TILE-Gx are provided by Ye Liu.

	Blackscholes	Streamcluster	Dedup	Canneal
Domain	Financial Analysis	Data Mining	Enterprise Storage	Engineering
Parallelization Model	Data Parallel	Data Parallel	Pipeline	Unstructured
Data Dependency	Low Data Sharing and Exchange	Low Data Sharing, medium Data Exchange	High Data Sharing and Exchange	High Data Sharing and Exchange
Number of Threads	1+n Threads	1+2n Threads	3+3n Threads	1+n Threads

Table 1: Features of Selected Benchmarks from PARSEC Benchmark Suite.

Blackscholes

Blackscholes is designed using simple structure of parallel programming, that the working threads are independent from each other. According to the description in technical report [9] and research [10], *Blackscholes* has its large percentage of execution time spent on the serial processing by the main thread, but not spawned worker threads for parallel processing. Also barrier synchronization is implemented for worker threads, which may worsen the serialization of the program.



(a) Intel Xeon Phi (b) TILE-Gx

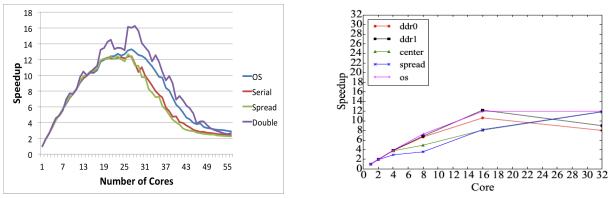
Figure 6: Scalability of *Blackscholes* on Both Platforms.

Scalability of *Blackscholes* on both platforms are shown in the Figure 6. *OS* and *Serial* patterns are measured for Blackscholes on Xeon Phi. Both platforms perform fair scalability when running *Blackscholes*, except that the severe serialization of the workloads result in reaching speedup saturation earlier on Xeon Phi than TILE-Gx. *OS* mapping shows almost the same scalability with *Serial* mapping on Xeon Phi, but on the other hand, the scalability shown on TILE-Gx can be improved by *OS* mapping. Bidirectional ring based Xeon Phi coprocessor is easier to reach its ceiling speedup than 2D-mesh based TILE-Gx processor.

Streamcluster

Streamcluster uses self-defined barrier implementation for synchronizing operations between worker threads. According to the research [11], scalability problem of Streamcluster is mainly because of the inefficient implementation of its self-defined barrier. Scalability of Streamcluster shown on both platforms are displayed in Figure 7.

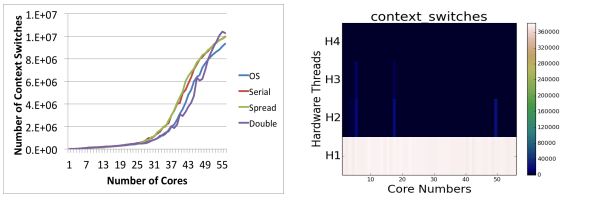
It clearly shows that Streamcluster scales far more better on TILE-Gx than Xeon Phi, after the designated threads number exceeds one half of the total number of cores. Speedup drops rapidly when number of designated threads are larger than 26 under *OS*, *Serial* and *Spread* mapping. *Double* mapping using hyperthreading technique improves scalability slightly because that two hardware threads share the same local L2 cache.



(a) Intel Xeon Phi (b) TILE-Gx

Figure 7: Scalability of *Streamcluster* on Both Platforms.

For further searching for the reason why Streamcluster performs so different between these two interconnect architectures, we measured the number of *Context Switches* and made the dynamic heat map for *Context Switches* happened on all the hardware threads of Xeon Phi. As shown in Figure 8(a), the increasing trend of *Context Switches* correlates with the scalability, which has shown in Figure 7(a). Figure 8(b) Shows the heat map of *Context Switches* when the designated number of threads is 56, under *Serial* mapping on Xeon Phi. Almost all the context switches are massively happened on the first hardware thread of each core, where each software thread is bound.



(a) Context Switches (b) Heat Map of Overall System

Figure 8: Numbers of Context Switches of Streamcluster Using Increasing Number of Threads and the Heat Map When 56 threads designated on Xeon Phi.

Why are these additional context switches happened? We measured the read misses of last level cache during the execution of Streamcluster on Xeon Phi. Two hardware events are recorded by accessing the PMU of each core:

- **L2_READ_MISS_CACHE_FILL**: Number of data read accesses which are satisfied by remote L2 cache after missing the local L2 cache.
- **L2_READ_MISS_MEM_FILL**: Number of data read accesses which are satisfied by the main memory after

missing the local L2 cache.

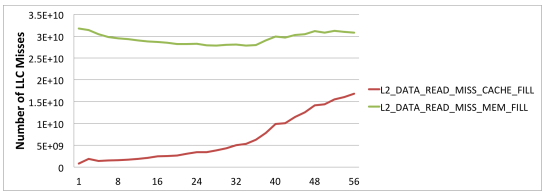


Figure 9: LLC read misses of *Streamcluster* under *Serial* mapping on Xeon Phi.

As the results shown in Figure 9, number of L2 load misses satisfied by remote cache increases significantly when designated number of threads is larger than 26. Larger values indicate that there are heavier traffics generated on the network. Network turns congested, especially on the smallest **AK** wire, when number of L2 load misses satisfied by remote caches is incremented. As the result that our self-designed micro-benchmark shows in Section 2.1, bidirectional ring is extremely sensitive to the network congestion. However mesh performs more flexible when deciding the routing path, therefore we conclude that the scalability difference on these two different architectures is also due to the network contention.

Dedup

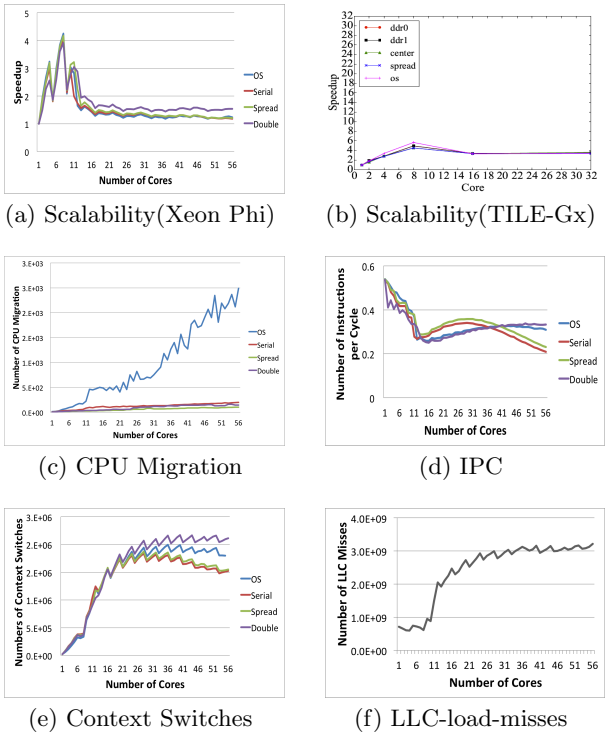
Dedup is selected because it uses pipeline parallelism and shows severe load imbalance between pipeline stages[12]. It has poor scalability on both platforms shown in the Figure 10.

Dedup reaches maximum speedup when 8 threads are used on both platforms. However Xeon Phi performs less competent when larger number of threads are spawned. The performance events *CPU Migrations* and *IPC* are summarized in the Figure 10. IPC decreases into 0.25 instructions per cycle on Xeon Phi when there are 56 threads designated by users. But according to the data collected on TILE-Gx, *IPC* keeps stable around 0.7 instructions per cycle. This indicates that Xeon Phi has more stall cycles and less instruction throughput during the execution of Dedup. Additional context switches also generate from LLC load misses during execution, which lead to heavy network congestion. Large number of CPU migrations happened under *OS* mapping indicates that the load-balancer of OS has done massive workloads between oversubscribed threads, and scalability is benefited a little from the load-balancer under OS mapping than Serial and Spread mappings.

The culprits of poor scalability on two experiment platforms are mainly due to their intolerance to severe load imbalance and intense competition for the shared resources between threads.

Canneal

According to the description in the report [9], Canneal has massive inter-core data exchanging and routing during its execution. We only show the scalability of Canneal on Xeon Phi here because the compiling of Canneal is unfortunately not successful on TILE-Gx. Hyperthreading technique unexpectedly deteriorates the average speedup into 1.5. Heat map of IPC under *Double* mapping when 56 threads spawned

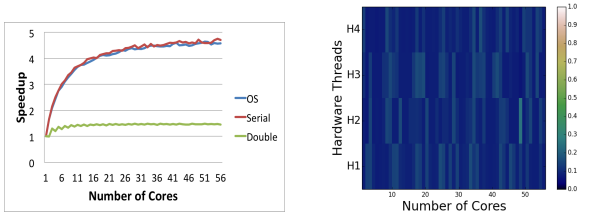


(a) Scalability(Xeon Phi) (b) Scalability(TILE-Gx)

(c) CPU Migration (d) IPC

(e) Context Switches (f) LLC-load-misses

Figure 10: Scalability and Performance Counters of Dedup on Intel Xeon Phi.



(a) Scalability

(b) Heatmap of IPC under *Double* mapping when 56 threads used

Figure 11: Scalability and IPC counter of Canneal on Intel Xeon Phi.

shows sluggish IPC counts on each hardware thread in Figure 11(b). We infer it's due to the competition of routers between two hardware threads on the same core. Further evaluation and analysis for this benchmark is going to be finished in the future work.

4. CONCLUSIONS AND FUTURE WORK

It is said that bidirectional ring usually performs worse than mesh network. We analyzed the main culprits of scalability bottleneck and explored the shortcomings need to be paid close attention during parallel programming for dedicated architectures. Although ring is more sensitive to network congestion, both networks are intolerant to severe load imbalance. Future works will be centered on OS kernel redesign and optimization for bidirectional ring and mesh based many-core processors.

5. REFERENCES

[1] Christian Bienia. *Benchmarking modern multiprocessors*. PhD thesis, Princeton University, January 2011.

[2] Intel Corporation. Intel Xeon Phi coprocessor system software developers guide. https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide, 2014.

[3] Rezaur Rahman. *Intel® Xeon Phi coprocessor architecture and tools*. Apress open, 2013.

[4] Jianbin Fang, Ana Lucia Varbanescu, Henk Sips, Lilun Zhang, Yonggang Che, and Chuanfu Xu. Benchmarking Intel Xeon Phi to guide kernel design. *Delft University of Technology Parallel and Distributed Systems Report Series, PDS-2013-005*, 2013.

[5] George Chrysos and Senior Principal Engineer. Intel Xeon Phi coprocessor (codename knights corner). In *Proceedings of the 24th Hot Chips Symposium*, 2012.

[6] Tilera Corporation. *Tile processor architecture overview for the TILE-Gx series*. USA, May 2012.

[7] Tilera Corporation. *Tile processor I/O device guide*. USA, 2011.

[8] Ye Liu, Hiroshi Sasaki, Shinpei Kato, and Masato Edahiro. A scalability analysis of many cores and on-chip mesh networks on the tile-gx platform. January 2016.

[9] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[10] Gabriel Southern and Jose Renau. Deconstructing PARSEC scalability. In *Proceedings of Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, 2015.

[11] Michael Roth, Micah J Best, Craig Mustard, and Alexandra Fedorova. Deconstructing the overhead in parallel applications. In *Workload Characterization (IISWC), 2012 IEEE International Symposium*, pages 59–68. IEEE, 2012.

[12] Angeles Navarro, Rafael Asenjo, Siham Tabik, and Calin Cascaval. Analytical modeling of pipeline parallelism. In *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*, pages 281–290. IEEE, 2009.