A Cost-Effective Load-Balancing Policy for Tile-Based, Massive Multi-Core Packet Processors

ENRIC MUSOLL
ConSentry Networks

Massive multi-core architectures provide a computation platform with high processing throughput, enabling the efficient processing of workloads with a significant degree of thread-level parallelism found in networking environments.

Communication-centric workloads, like those in LAN and WAN environments, are fundamentally composed of sets of packets, named flows. The packets within a flow usually have dependencies among them, which reduce the amount of parallelism. However, packets of different flows tend to have very few or no dependencies among them, and thus can exploit thread-level parallelism to its fullest extent.

Therefore, in massive tile-based multi-core architectures, it is important that the processing of the packets of a particular flow takes place in a set of cores physically close to each other to minimize the communication latency among those cores. Moreover, it is also desirable to spread out the processing of the different flows across all the cores of the processor in order to minimize the stress on a reduced number of cores, thus minimizing the potential for thermal hotspots and increasing the reliability of the processor. In addition, the burst-like nature of packet-based workloads render most of the cores idle most of the time, enabling large power savings by power gating these idle cores.

This work presents a high-level study of the performance, power, and thermal behavior of tile-based architectures with a large number of cores executing flow-based packet workloads, and proposes a load-balancing policy of assigning packets to cores that minimizes the communication latency while featuring a hotspot-free thermal profile.

Categories and Subject Descriptors: C.1.4 [**Processor Architectures**]: Parallel Architectures—*Distributed architectures*

General Terms: Design, Performance, Reliability

Additional Key Words and Phrases: Multi-core, many-core, packet processing, load balancing

ACM Reference Format:

Musoll, E. 2010. A cost-effective load-balancing policy for tile-based, massive multi-core packet processors. ACM Trans. Embedd. Comput. Syst. 9, 3, Article 15 (February 2010), 25 pages.
DOI = 10.1145/1698772.1698782 http://doi.acm.org/10.1145/1698772.1698782

Author's address: E. Musoll, ConSentry Networks, 1690 McCandless Drive, Milpitas, CA 95035; email: enric@consentry.com.

1. INTRODUCTION

Multi-core architectures are becoming the de facto computing model for applications (such as networking, DSP and e-commerce) that present a high degree of thread-level parallelism. These workloads are executed efficiently in multi-core architectures, in terms of both power and performance: power can be significantly saved by power gating unused cores, and performance increases almost linearly with the number of cores as long as enough thread-level parallelism exists. Future multi-core processors are expected to feature hundreds of cores thanks to the increasing transistor integration capacity of future process nodes [Borkar et al. 2007].

The main metrics that define how suitable a particular multi-core architecture is to execute a certain workload are the following.

- —Performance. Performance is defined by both the number of processed requests per second, that is, the processing throughput, and the processing latency per request.
- —Power. Massive multi-core architectures are composed of a large number of small cores usually interconnected among them in a mesh-like fabric. This pool of cores potentially presents a high power density profile when the majority of the cores are running. But due to the burst-like nature of the application workload [Jiang and Dovrolis 2003], most of the time several cores are idle, and therefore they can be either clock gated (thus saving dynamic power) or power gated (virtually saving all the power, but at a more costly design effort). In any case, the regular nature of massive multi-core architectures coupled with the burst-like nature of the workloads, makes these type of architectures very appealing to power saving techniques.
- —Temperature. The high-power density of massive multi-core architectures translates into a high die temperature. The maximum temperature that can be reached when all the cores are running may be above the maximum temperature that the heat dissipation mechanism in place (package, heat sink, air flow) can absorb. In this case, the operating system or some special-purpose hardware takes action by throttling the ingress of requests or lowering the power supply and/or frequency. In any case, even a low rate of requests can create a hot spot if the cores assigned to the requests are close to each other and the same set of cores is used all the time. In this case, even though the overall power consumption of the processor is not high, the fact that the power density is concentrated in a portion of the die creates some problems:
  - (a) The design of the package needs to accommodate these localized hot spots, thus increasing its cost;
  - (b) If the localized hot spot always occurs in the same part of the die, implying that the same cores are always being used, the wear of the chip becomes imbalanced, thus increasing the probability of failure due to the presence of heavily stressed cores.

Communication-centric workloads are fundamentally composed of sets of packets, named flows. A flow is a sequence of packets from one particular source to a single destination. More specifically, the packets of a flow share the same 5-tuple set of fields (source/destination address, source/destination port and protocol type), effectively linking them to the same application data set, and therefore, forcing a certain level of communication and/or synchronization among them.

The degree of statefulness (data context shared among the packets of the same flow) is higher in the upper layers of the OSI reference model (for example the computation of the flow virus signature [Keromytis and Prevelakis 2007]) than in the lower ones (for instance the next hop information in the routing table). In any case, this degree of statefulness reduces the amount of thread-level parallelism among the packets of the same flow. However, packets of different flows tend to have almost no dependencies among them, and can fully exploit the thread-level parallelism.

Therefore, in multi-core architectures, it is important that the processing of the packets of a particular flow takes place in a set of cores physically close to each other to minimize the communication latency among them. This is especially important in tile-based multi-core architectures, which present a mesh-type interconnect fabric that provides high communication bandwidth, low area overhead, a very regular tile-like floorplan and a higher energy efficiency when compared to a bus-based topology [Konstantakopoulos et al. 2007], but at a cost of increased communication latency when the cores are physically distant from one another. Other network topologies for multi-core such as the concentraced mesh [Balfour and Dally 2006] and flattened butterfly [Kim et al. 2007] have been proposed to improve the limitations of the mesh interconnect. In this work, however, we assume a simple mesh interconnect similar to the ones used in recent tile-based multi-cores [Agarwal et al. 2006; Vangal et al. 2007].

Flow-based load-balancing schemes in multi-core architectures with a small number of cores usually map a particular flow to a single core, so that the processing order of packets within that flow is preserved, and the temporal locality of the data associated to the flow is exploited (as long as the flow aggregation degree is low) [Shi et al. 2003]. A problem with these schemes is the workload imbalance that can occur in the presence of dominating flows [Shi et al. 2005]. Moreover, extra buffering and control logic are needed to absorb incoming bursts of packets corresponding to the same flow.

In multi-core architectures the flow data locality can be preserved as long as the architecture features a distributed, shared level of caching [Azimi et al. 2007; Muralimanohar and Balasubramonian 2007]. Cores processing packets of the same flow need however to synchronize to maintain the proper packet processing ordering within a flow.

The load balancing goal for massive tile-based multi-core architectures then is to use as many cores as the ingress bandwidth and application workload requires, with cores that simultaneously process packets of the same flow physically close to each other. The reduced communication latency among packets of the same flow decreases the overall flow processing time. This is desirable for these reasons.

—Lower packet processing latency. Since the core will spend less time waiting for remote data, it will complete the processing faster.

—Higher processor processing throughput. When most of the cores of the processor are actively executing packets, the longer it takes to process a packet the lower the overall processor throughput is. This is not the case when several of the cores are idle, where longer processing time is traded off for more active cores, so the overall throughput remains the same (however, more power is dissipated since more cores are active in average).

—Lower power dissipation. As soon as a core finishes processing a packet, it can transition into a lower power state.

A second objective of the load-balancing policy is to spread the processing of the different flows across all the cores of the processor in order to minimize the stress that would otherwise occur if the same reduced set of cores is always used to process packets. A proper load balancing of the packets across the cores reduces the potential for thermal hotspots and increases the reliability of the processor.

This work presents a high-level study of the performance, power and thermal behavior of tile-based multi-core architectures (featuring a mesh-based interconnect fabric) executing flow-based packet workloads, and proposes a load-balancing policy of assigning packets to cores that minimizes the communication latency while featuring a hotspot-free thermal profile. The idea behind this policy is to dynamically restrict, based on the number of flows being processed, the preferred set of candidate cores to process an incoming packet of a particular flow. This dynamic restriction increases the likelihood of two packets from the same flow to be processed in nearby cores, and enables a good thermal hotspot behavior by spreading as much as possible the processing of flows across all the cores.

2. DYNAMIC VIRTUAL CLUSTERING

The basic idea behind the proposed load balancing policy, henceforth named Dynamic Virtual Clustering (DVC), revolves around the concept of a virtual cluster. Figure 1 illustrates this notion. The top left diagram shows a 4x4 tile-based multi-core, where cores are numbered starting from the lower left corner, and increasing on a row-first basis. This diagram shows all the 16 cores being part of a single virtual cluster (VC#0). This corresponds to a virtual cluster level (VCL) of 0. The higher the virtual cluster level, the larger the number of virtual clusters (2^{VCL}). Virtual clusters are numbered following the same convention as cores within a cluster (start at lower left corner, proceed row first), as the rest of the diagrams in Figure 1 show. Note that the figure does not offer the case of VCL = 4; this case corresponds to 16 virtual clusters, the same number of physical cores.

The key decisions that the DVC policy has to address are:

—which virtual cluster the incoming packet will be assigned to (and to which core within that cluster), and

—when to switch to a different virtual cluster level (VCL).

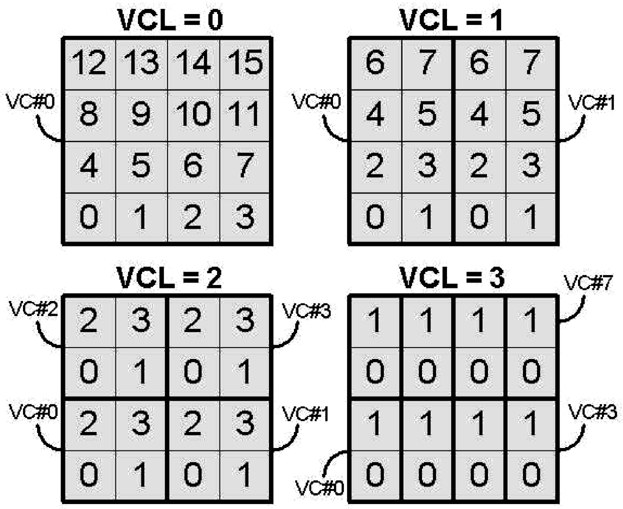


Fig. 1. Virtual clustering concept.

2.1 Virtual Cluster Number Selection

Packet processing systems can handle a limited number of active flows. Once the system detects that the incoming packet does not belong to any of the currently active flows, a new flow is created (if the limit has not been reached) and a flow number (typically generated as a hash of several fields of the packet) is assigned to that flow. From that point on, and until the last packet of the flow is processed (or the flow ages out due to inactivity), the flow remains active.

The DVC policy follows a simple mechanism to assign packets of a particular flow to a virtual cluster: the VCL least significant bits of the flow number associated to a pac ket determine the virtual cluster number. The maximum value of VCL is $2\times \log_2(N)$, where N is the number of cores on the side of the processor (for a total of N×N cores).

Assuming that the flow number has been generated with a high-quality hash function,[1] the contents of these bits provide a uniform distribution that translates into all the clusters being regularly used. This in turn helps spreading the workload across the chip, which increases the reliability of the processor by not stressing a small portion of the die. Once the virtual cluster is selected, a lower-index-core scheme is used to assign the packet to a core within the cluster (i.e., the available core with the lowest index with in the cluster is selected). In the case that a packet has to be assigned to a particular virtual cluster but all the cores within that cluster are busy executing other packets, DVC attempts to find a core in another virtual cluster (rather than buffering the packet until a core in the desired cluster is available), following a simple round-robin strategy on virtual clusters until one is found with at least one available core.

[1]Both the checksum and CRC algorithms over the 5-tuple provide a good random beha vior [Jain 1992].

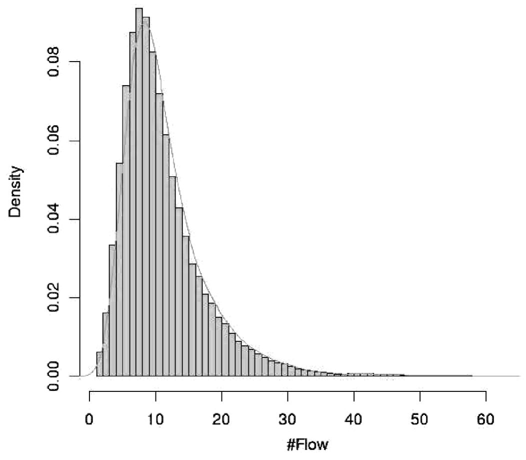


Fig. 2. Probability density of the number of active flows in a particular multi-core system.

2.2 Virtual Cluster Level Selection

As mentioned before, a packet processing system maintains up to a fixed number of total active flows (TAF). An active flow is defined as a flow whose first packet has been seen by the system (so the flow has been assigned a flow number), and its last packet still has to be processed.

Depending on how the aggregation of the different flows has taken place prior to the arrival at the packet processing system, on the burst-like nature of each flow, and on the particular characteristics of the application, packets of only a subset of those active flows are being executed by the cores of the processor. It may be the case that TAF is large (a million or more in state-of-the-art routers), but packets of only a handful of those flows are being processed during a certain period of time. In other words, processing of flows may present temporal locality. Shi et al. [2005] show that the number of active flows, or flows in transit, varies significantly as it can be derived from the probability density plot in Figure 2 (reproduced from Shi et al. [2005]). It is expected, then, that the number of flows having packets being processed during a period of time also varies over time.

The DVC policy selects the level of virtual clustering (VCL) based on the number of active flows that have packets being processed by the cores (henceforth referred to as currently running flows or CRF). More specifically,

$$VCL = min\left(2^{\lceil \log_2(CRF) \rceil}, NC\right), \tag{1}$$

where NC is the total number of cores available in the processor. The total number of virtual clusters can not exceed the number of physical cores. Therefore, as the number of currently running flows increases, so does the number of virtual clusters, but at a power-of-two granularity, as was illustrated in Figure 1.

Figure 3 provides an example of how the DVC technique works for a simple scenario involving four different flows with an arbitrary number of packets and processing times and compares the load balancing decisions with two straightforward cluster-agnostic techniques.

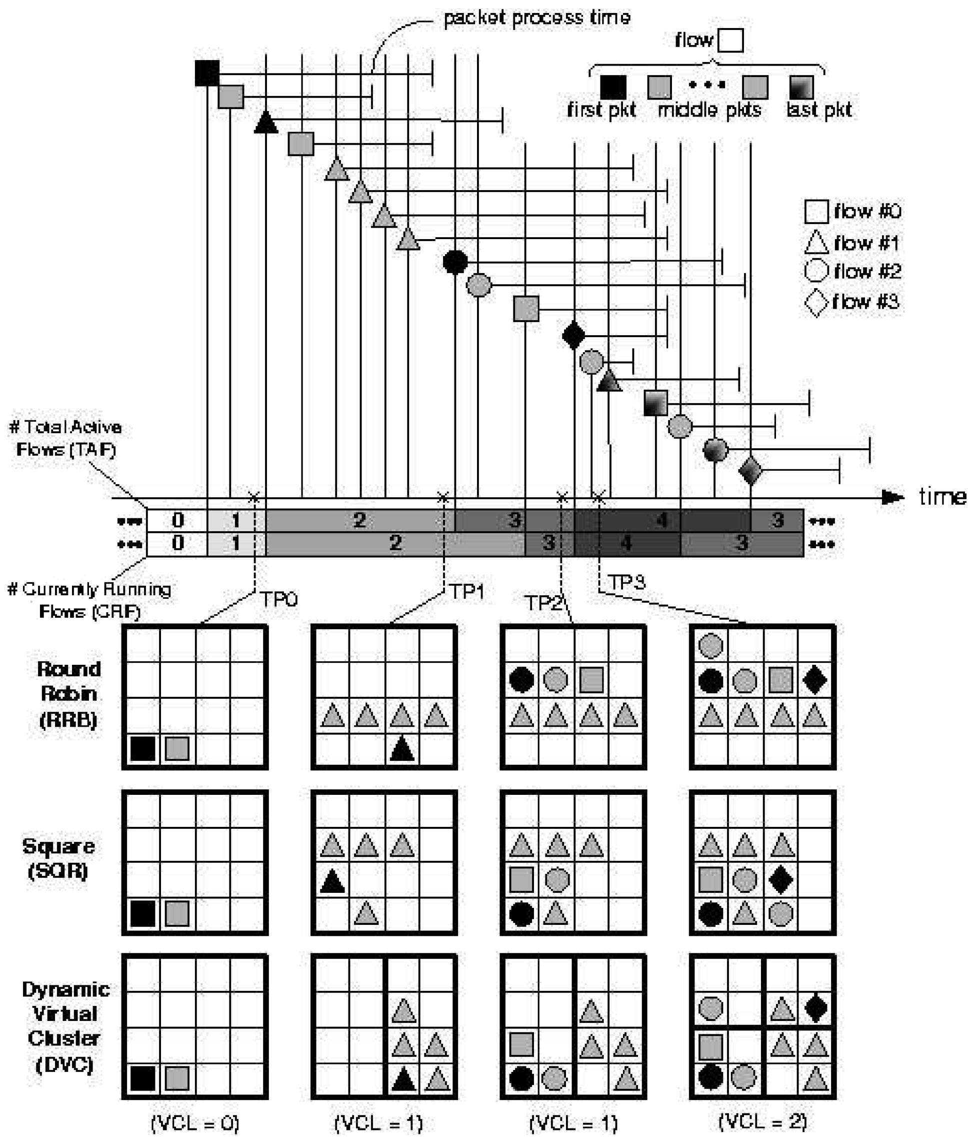


Fig. 3. Load balancing example for Round Robin, Lower Index First, and Dynamic Virtual Cluster.

—Round Robin (RRB). The next core is selected in a round-robin fashion.
—Square (SQR). Cores are selected so they concentrate on the same reduced area (lower left corner of the die).

The figure shows which packets are being processed by which cores at four specific time points (TP0. . . 3) for the three different techniques. The total number of active flows (TAF) and the number of currently running flows (CRF) are also shown. Note that in this example TAF = CRF except for two time intervals, where CRF is smaller. The first time point occurs when CRF = 1, the second when CRF = 2, and so on, which translates to a VCL value of 0, 1, 1, and 2 respectively. The following observations help to clarify the dynamics of the DVC load-balancing mechanism.

—Upon receiving the first packet of the trace (flow #0), CRF is set to 1 since only one flow has packets being processed once this packet is assigned to a core. Since CRF is 1, VCL becomes 0, and a single virtual cluster is defined.
—The first two packets of the trace are assigned to the single virtual cluster. This is the status at TP0.
—Upon receiving the first packet of flow #1, DVC recalculates CRF to 2 because packets of two different flows will be concurrently processed once this packet is assigned to a core. VCL becomes then 1, leading to two virtual clusters from now on. Since the LSB bit of flow #1 is 1, the packet is assigned to VC#1.
—The next packet in the trace (second packet of flow #0), is assigned to VC#0 since at the time of its arrival, VCL is 1 and the LSB bit of flow #0 is 0. Similarly, the next 4 packets (of flow #1) are assigned to VC#1. At TP1, the two packets of flow #0 have already completed their processing, while all the packets received from flow #1 are still being processed. Note that flow #0 is still active at the system level since its last packet has yet to be seen, even though no packet from flow #0 is being processed at TP1. Also note that even though at TP1 there are only packets of a single flow being processed, CRF remains at 2 because the update of CRF only happens at a start-of-processing event, not at an end-of-processing event.
—The next three packets all are assigned to VC#0 since they belong to either flow #0 or #2. The first packet of flow #2 increases TAF to 3. However, CRF remains at 2.
—Upon the arrival of the first packet of flow #3, both TAF and CRF are updated to 4 and VCL changes to 2. This packet is then assigned to VC#3, as indicated by the 2 LSB bits of its flow number.
—At time TP3 there are 4 virtual clusters. Note that packets of flow #1 spread across more than one virtual cluster. This is allowed since the number of virtual clusters changes dynamically and the processing of packets do not migrate to another core once they have started their processing. Incoming packets of flow #1 will be assigned to VC#1, and the packet in VC#3 will remain there until it completes the processing. Similarly, the packet from flow #2 arriving right before TP3 was assigned to VC#2, while other older packets of the same flow are running on VC#0.

The core assignment snapshots in Figure 3 for RRB and SQR reveal the expected behavior of packets being scheduled (irrespectively of their associated flow number) to either the next contiguously available core (RRB) or the first available core in the lower left corner of the chip.

3. FRAMEWORK ENVIRONMENT

In this work we target future massive multi-core architectures processing flow-based packet workloads. These architectures, featuring hundreds of cores,

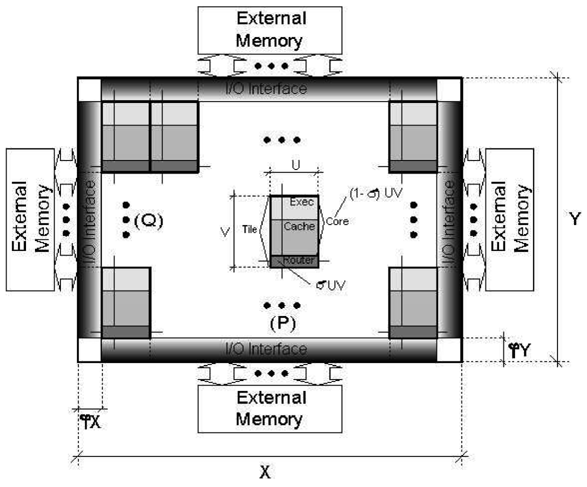


Fig. 4. Generic tile-based multi-core architecture.

usually present a mesh-type interconnect fabric. Some early examples of these types of architectures are Agarwal et al. [2007] and Hoskote et al. [2007]. Future implementations are expected to integrate a significant number of additional cores thanks to the increasing area density of future technology nodes, the inherent scalability of mesh-based interconnect fabrics and the design-friendly tile-based floorplan.

In this section we describe the area, performance and power model of a generic tile-based multi-core architecture, and the packet activity generator used in this work to exercise the processor model toward the evaluation of the proposed load-balancing technique. This model is tailored for mesh-based interconnection fabrics, which are the most cost effective way to connect a large number of cores.

3.1 Area Model

Figure 4 shows a diagram of a generic implementation of a tile-based homogeneous multi-core processor, with external memory interfaces in all four sides.[2] Each tile consists of two main blocks: the core (execution pipeline plus cache/memory) and the router, which connects the tile with the neighboring tiles via four links (north, south, east and west). In this work, we assume that each of these links can support one code or data request per cycle in each direction. The parameters shown in the figure are described as follows.

—X, Y are the processor die dimensions.
—P, Q are the number of tiles on the side. The total number of cores is then $P \times Q$.
—U, V are the core dimensions.

[2]We do not show the packet interface nor the special-purpose hardware that performs the buffering and load balancing of the packets. Our focus is to concentrate on the performance, power and thermal behavior of the pool of cores.

—φ is the width fraction of the I/O Interface block, with respect to X and Y.
—σ is the fraction of the core area that is used by the router.

The figure offers a core floorplan (a stack of the three subblocks) and a top-level floorplan (tiles side by side in a regular 2D layout with no abutment). On each side of the die, the I/O Interface block bridges the requests from the periphery routers to external memory. This floorplan is assumed throughout this work.

3.2 Performance Model

The time $T(r, c)$ it takes to process a request r on a core c is modeled as follows:

$$T(r, c) = \frac{I(r) \cdot ((1 - \alpha(r)) \cdot Z + \alpha(r) \cdot M(c))}{F},$$

where:

—$I(r)$ is the total number of instructions to be executed to process request r.
—$\alpha(r)$ is the fraction of the instructions for request r that are memory related.
—Z is the average number of cycles it takes to execute a nonmemory instruction, including data and control dependencies, by a core. This number is the same for all cores.
—$M(c)$ is the average memory latency in cycles for a memory instruction executed in core c.
—F is the operational frequency of a tile.

This expression assumes an in-order execution engine, which is representative of the small, simple cores that massive multicore architectures feature.

The average memory latency $M(c)$ takes into account the accesses to both local and remote memories. A remote memory access may be served by another core or by the external memory. $M(c)$ is defined as follows.

$$M(c) = (1 - \beta) \cdot L + \beta \cdot R(c)$$

$$R(c) = \frac{\sum_{k \in \Phi(c)} \{2 \cdot H(c, k) + L\} + (2 \cdot H(c, \bullet) + E)}{|\Phi(c)| + 1},$$

where:

—β is the fraction of memory instructions that is served remotely (either by another core or by external memory).
—L is the latency in cycles to obtain local data. This latency is the same for all cores.
—$R(c)$ is the average latency in cycles to obtain remote data by the core c.
—$\Phi(c)$ is the set of cores (other than c) that own data that core c will eventually access. Let Ω be the set of cores that are running at any given time. So, if a core c accesses the memories of all other running cores, then $|\Phi(c)| = |\Omega| - 1$.

—$H(c, k)$ is the Manhattan distance, in number of tiles (or hops), between core c and core k. If the two cores are side by side, this distance is 1. $H(c, \bullet)$ refers to the Manhattan distance between core c and the external memory.
—E is the latency of the external memory, including the on-chip I/O interface.

Implicit in this expression are the following assumptions.

—The fraction of remote memory accesses (β) is equally distributed among the other running cores that own some of the requested data (Φ set), and the external memory. For instance, when no other core is running, then the whole β fraction of memory accesses is served by the external memory; if there is a single core running that owns some of the requested data, $\beta/2$ fraction of memory accesses is served by the other core's local cache and the other $\beta/2$ fraction is served by the external memory.
—It takes one cycle to traverse a tile. Therefore, if a core c accesses the memory of another core k, the latency of routing the request, and later on the response, is $2 \cdot H(c, k)$ plus the memory access latency itself.[3]
—The portion of remote accesses to external memory is assumed to be equally divided among the four physical sets of memories (one in each side of the processor). Therefore, the average distance of core c to external memory ($H(c, \bullet)$) is actually $(Q + P + 2)/4$ regardless of the placement of the core in the die.
—The model assumes no contention in the router blocks of the different cores. As it will be shown later, contention in these architectures is in average very low.

Moreover, we assume that once a core has finished processing a request, it writes back any dirty lines in its data cache and then invalidates all valid lines. Thus, a core that is not executing any request does not serve any remote access performed by any other core.[4]

In mesh-based, massive multi-core architectures, contention on the router blocks of the tiles is in average low. The main reasons are the following.

—A mesh-based interconnect fabric presents a high bandwidth mechanism for data transfer. In this work, we assume that each tile can sustain five requests per cycle (to support the core's own remote accesses and to serve other core's remote accesses or to route them to the proper destination).
—The cores tend to be simple, in-order, nonspeculative engines that issue memory requests at a rate lower than its peak IPC. So, assuming a single-scalar

[3]Actual implementations of routers could take at least one cycle but usually three [Dally and Towles 2003; Peh and Dally 2001; Mullins et al. 2004] plus an additional one for the link to the neighbor tile. A small tile traversal latency minimizes the remote access latency, favoring those load-balancing techniques that assign packets of the same flow to spread out cores. This is not the case for the technique proposed in this work, so the single cycle tile traverse latency is a conservative assumption when comparing the proposed technique to the other ones.
[4]We made this assumption to simplify the performance model. The alternative of leaving the data in the cache requires to keep the cache powered up when the core is idle, plus it introduces an additional parameter to the model to account for data cache pollution across different flows.

core, this rate is never higher than one request per cycle per core. Moreover, several cores are often idle due to the burst-like nature of the workload, further reducing the overall injection rate of remote traffic into the mesh fabric.

—Not all the instructions executed are memory or synchronization requests. In typical workloads, around 50% or less of the instructions need to access memory. Moreover, depending on the application, a large portion of the memory requests is locally served within the tile. Assuming a 20% local cache miss rate ($\beta = 0.2$) and 50% of memory instructions in the workload ($\alpha = 0.5$), and a single-scalar core ($Z \geq 1$), the rate of remote accesses is at most one every four cycles.

Assuming each link can take one request per cycle, contention in a tile's router can occur when two or more requests are ready to be sent to the same link at the same time; only one is able to proceed, and the other(s) need to wait in link buffers. To illustrate the previous claim that the contention is low, the following simulation is conducted.

—$P = 16$ and $Q = 16$, for a total of 256 cores.
—$Z = 1$ and $L = 1$; thus, the peak IPC is 1, achieved when all memory requests are local.
—$E = 200$, which at a reasonable tile's 1-2GHz frequency range, amounts to 200-100ns latency, including the latency at the I/O Interface blocks of the processor.
—$|\Phi(c)| = |\Omega| - 1$, implying that a core accesses all the other currently running core's memories.
—The remote accesses by a core are equally distributed between (a) all the other running cores processing packets of the same flow, and (b) the external memory.
—The accesses to external memory are equally distributed among the four sets of external memory chips.
—The size of the requested data corresponds to the physical width of the link (thus the transfer of data between neighboring tiles takes a single cycle).
—The external memory chips have enough ports to support the requests generated by the processor. In other words, the external memory bandwidth is not the bottleneck.

Figure 5 shows the experiment results in terms of overhead percentage with respect to the no-contention case. The plot presents this overhead for a range of running cores ($|\Omega| = \{16, 32, 64, 128, 256\}$) and several values of remote access percentages ($100 \cdot \beta$). We can observe that the worst case in contention occurs, as expected, when all the cores are running and when all the memory instructions are serviced by a remote core or external memory. But even in this extreme scenario, the overhead is less than 5% over the case in which no contention exists. For more realistic scenarios, like for example $\beta = 0.1$ and $|\Omega| = 128$, the overhead is about 1%. A simple store-and-forward, shorter-distance, column-first routing algorithm is used in the experiment.

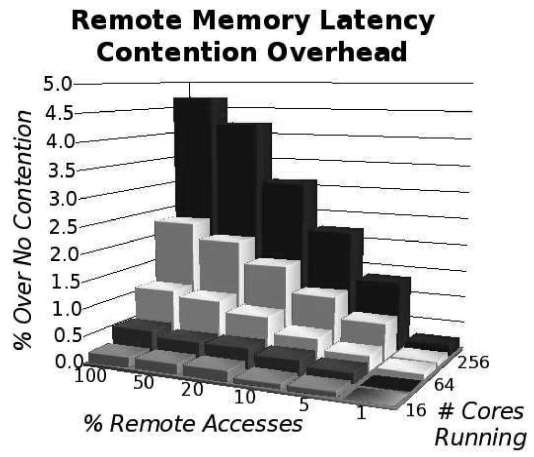


Fig. 5. Contention overhead experiment ($P = Q = 16$, $Z = 1$, $L = 1$, $E = 200$, $\alpha = 0.5$).

These results correlate well with the work in Konstantakopoulos et al. [2007], where an analytical study (validated with real traces) is made for a 4x4 tile-based processor. In that study it is shown that an injection rate of around 6% provides approximately a 3% probability of contention. When we apply our model to this case ($\alpha = 0.5$ and $\beta = 0.12$ for an injection rate of 6%, and $P = Q = 4$), we obtain 1.9% overhead. Therefore, in this work we do not factor in any contention penalty on remote accesses since it has been demonstrated to be small.[5]

3.3 Power Model

Power consumption of a block is composed of a dynamic component (which depends on the voltage supply, the clock frequency, the capacitance, and the signal activity), and a static component (which mainly depends on the voltage, temperature and capacitance). Henceforth, let the static portion ratio of the total power be δ.

We divide the power consumption of a tile into two values: the core's power and the router's power. Furthermore, we express the router power as a fraction of the tile's power as follows:

$$W_{tile} = W_{core} + W_{router} = W_{core} + \lambda \cdot W_{tile} = \frac{W_{core}}{1 - \lambda},$$

where W_{router} is the average power of the router when all its links are being used all the time, W_{core} is the average power of the core when it is running and λ is the router share of the overall tile power.

Power gating [Roy 1998] is a very attractive technique to reduce power consumption in multi-core architectures due to not only the opportunities that the burst-like workload presents, but also to the layout regularity of the architecture: the power gating circuitry can be designed for a single core, and

[5]Again, no contention on the routers favors the techniques that assign packets to distant cores, which is not the case with the proposed technique.

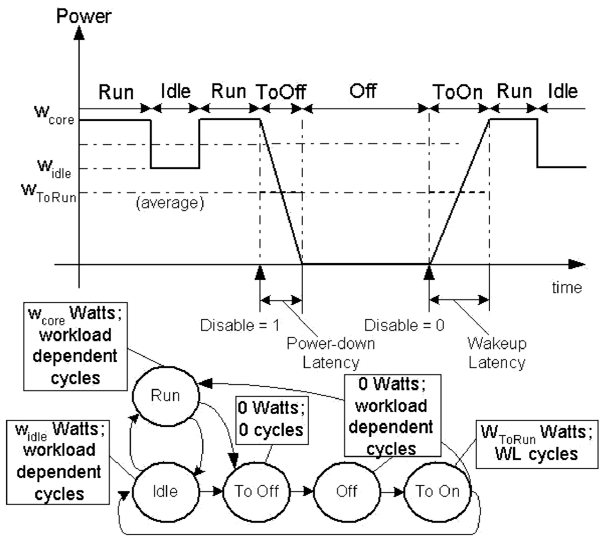


Fig. 6. Power gating states and state transition diagram modeled.

deployed as many times as cores exist. The different cores in the processor can be switched on and off by using, for example, a properly sized header (footer) switch per core, composed of several transistors. This mechanism allows a particular core to be shut down when it is not executing anything and woken up when needs to execute a request. When the core is shut down, virtually no power is dissipated.

There are however both power and performance penalties when waking up a core: the core needs to stabilize its voltage source (ground rail) before starting the processing, there is extra power spent on the header (footer) circuitry due to the gating of relatively large transistors plus the buffering of the on-off signal from the control logic to the header (footer) switch. For a more in-depth description of the different phases that exist in powering on/off a circuit, refer to Hu et al. [2004].

We model the power gating mechanism as follows (see Figure 6).

—Whenever the core is running, it consumes W_{core} Watts.
—Whenever the core is waiting for remote data, it is clock gated, thus consuming only static power ($(1 - \delta) \cdot W_{core} = W_{idle}$ Watts)
—Whenever the core has completed the processing of the packet, it transitions instantaneously into the off state, consuming no power. Power gating effectively eliminates the power consumption in the gated circuit, except for the small contribution of mainly the header/footer switch. Any power and latency overhead that this transition involves is lumped into the off-to-run transition (W_{ToRun} Watts and a wakeup latency penalty of WL cycles).

Moreover, power gating is applied only to the core and not to the router block since it is needed to route accesses even if the tile's core is not processing any packet. Power gating the routing block incurs in a large performance penalty because it needs to be woken up to route a request. Adding ten cycles or more of wakeup penalty to the one-cycle routing task is a very significant overhead that heavily impacts the performance of the core that issued the remote access. Thus, the router block r always consume static power, defined as follows:

$$W_{router}^{sta}(r) = \delta \cdot \lambda \cdot W_{tile}$$

and, the dynamic power is proportional to the routing activity as follows:

$$W_{router}^{dyn}(r) = (1 - \delta) \cdot \lambda \cdot W_{tile} \cdot \frac{P(r)}{P_{max}(P, Q, \alpha, \beta)},$$

where $P(r)$ is all the possible paths that cross router r and $P_{max}(P, Q, \alpha, \beta)$ is the theoretical maximum number of these paths (which depends on the number of cores and the injection rate of remote requests). $P(r)$ depends on the number of running cores (including the core in the same tile as router r) and whether remote requests by these cores are serviced or routed to other tiles by router r. For $P = Q = 16$ and all cores injecting requests every cycle (except when the core is waiting for the requested data), P_{max} is found experimentally at $25,088$.

The power of the I/O block interface ($W_{I/O}$) is set to a fixed, constant value since it is expected to always be active scheduling the external memory requests from the tiles.

Armed with both power and performance estimates for running a request in a particular tile, a load-balancing mechanism (implemented in hardware or at the OS level) can perform a quick decision on which of the idle cores the request will be executed. A fast decision is desirable when executing light workloads (of 100's instructions, as in some networking applications), where the load-balancing decision latency overhead can be significant.

3.4 Traffic Model

We use a synthetic packet traffic generator to exercise the tile-based multi-core architecture described above. This traffic generator creates packets at a specific rate (PKR) in millions of packets per second. There can be up to TAF$_{max}$ number of active flows in the system, and up to CRF currently running flows (CRF $\leq$ CRF$_{max}$ $\leq$ TAF $\leq$ TAF$_{max}$) in the processor. The minimum/maximum number of packets per flow is PPF$_{min}$/PPF$_{max}$ (uniformly distributed).

An important knob of the traffic generator is the packet burst length (PBL), which specifies how many packets from the same flow will be consecutively generated. When PBL is 1, the packet generated corresponds to a flow number different from the previous one (unless of course CRF is 1). PBL can be larger than PPF for a particular flow, in which case the burst length for that flow will be PPF (ie all the packets in that flow will be generated consecutively).

Figure 7 illustrates how the generator works with a simple example where TAF$_{max}$ = 4, CRF$_{max}$ = 2, PBL = 3, PKR = 1, PPF$_{min}$ = 3 and PPF$_{max}$ = 6. We can observe that the periods of flow locality vary in length (even though CRF$_{max}$ and PKR have fixed values) because of the variability in the number of packets per flow. Moreover, whenever a new flow is created, it is assigned an available, random flow number (0 ... TAF$_{max}$−1).

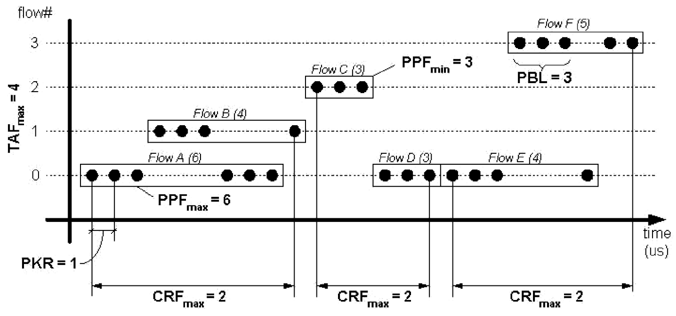


Fig. 7. Example of traffic trace generation for TAF$_{max}$ = 4, CRF$_{max}$ = 2, PBL = 3, PKR = 1, PPF$_{min}$ = 3 and PPF$_{max}$ = 6.

Table I. Description of the Model Parameters and Values Evaluated

	Area Parameters	
X, Y	die x,y size	20,20 mm
P, Q	#tiles on x,y side	16,16
U, V	tile x,y size	1.2,1.2 mm
σ	router/tile size ratio	0.3
φ	I/O i/f width vs X, Y ratio	0.02
	Performance Parameters	
$I(r)$	#inst. per request r	(1K,10K,100K)
$\alpha(r)$	memory/all inst. ratio in $I(r)$	0.4
β	remote/all mem. access ratio	(0.3,0.5)
$Z(c, r)$	avg. clocks per non-memory instr.	1
L	#cyc. of rout. lat. to local data	1
E	#cyc. of rout. lat. to external data	100
F	tile clock frequency	1GHz
WL	core wakeup latency in cycles	100
	Power Parameters	
δ	static/total power ratio	(0.2,0.6)
λ	router/tile power ratio	0.28
W_{core}	core power when running	180 mW
W_{ToRun}	core power when waking up	90 mW
$W_{I/O}$	total I/O i/f power	5 W
	Traffic Generator Parameters	
PKR	incoming packet rate	(0.5,1,2,4) MPPS
TAF$_{max}$	max total active flows	100K
CRF$_{max}$	max currently running flows	(10,50,100)
PBL	packet burst length	(1,4,16)
PPF$_{min}$	min number of packets/flow	10
PPF$_{max}$	max number of packets/flow	100

3.5 Summary of the Models

Table I summarizes the parameters that we have previously presented, and provides the values that will be used henceforth. A thorough study of each of the parameters is beyond the space constraints of this work, so we provide reasonable values for most of them and focus in more detail on the rest.

In particular, we focus on an aggressive die size (20×20 mm^2) and number of cores (16×16 = 256), each core with a base core power of 180 mW and router power of 70 mW ($\lambda = 0.28$, based on the reported value of 28% in Hoskote et al. [2007]). The wakeup penalty of a core is set to 100 cycles, a reasonable number for a relatively complex block with no state retention [Keating et al. 2007]. The tile frequency is set to 1 GHz.

The static power ratio δ is set to 0.6, which has been typically reported as representative of current and near future technologies [Kim et al. 2003]. However, in a 65 nm implementation of a high-performance mesh interconnect [Hoskote et al. 2007], the authors report a leakage power of around 15% the overall power. Therefore, we also evaluate the case of $\delta = 0.20$ to model advanced processes featuring techniques like high-k dielectrics to reduce the gate-oxide leakage portion of the static power.

The router block area is modeled to be 30% of the tile area ($\sigma = 0.3$), based on tile layout measurements in Hoskote et al. [2007] and in Balfour and Dally [2006]. The I/O interface width ratio (φ) is set to 0.04, rendering the total I/O area to 64 mm^2 (8% of the total die).

The number of instructions to execute per packet ($I(r)$) is set to 1K, 10K or 100K, with 40% of them accessing memory, out of which 20% or 50% require remote accesses.[6]

Finally, for the traffic generator model, we generate four ingress packet rates (0,5, 1, 2 and 4 MPPS), 100K active flows, three maximum currently running flows (10, 50 or 100), packet burst lengths of 1, 4, or 16, and 10 to 100 (randomly distributed) packets per flow.

4. RESULTS

In this section we present some of the results obtained with our event-based simulator that implements the framework environment previously described. Due to space constraints, we focus only on a relatively small set of results. A comprehensive study of all the design space is beyond the scope of this work.

Figures 8 and 9 plot the average number of cores, the processing latency per packet, the overall processing throughput and the power consumption when varying the ingress packet rate and remote access ratio (x-axis), and the number of instructions per packet (each row of plots). The packet burst rate is set to 1 and the maximum currently running flows is fixed at 10. Later on we will analyze the effect that these two parameters have as they take different values. For the power plots, we also distinguish between a static power ratio of 20% and 60% (last two plot columns). The three load-balancing techniques discussed in this work (RRB, SQR and DVC) are evaluated.

Some conclusions that can be derived from these results are the following.

Regarding Performance.

—For a given remote access ratio, the latency decreases as the ingress rate increases. The reason stems from how we model the communication among cores: the larger the number of simultaneously running cores processing packets of the same flow, the larger the percentage of remote accesses will be

[6]Therefore, we basically model the synchronization/communication among cores simultaneously running packets of the same flow as a percentage ($\alpha \cdot \beta$) of all the instructions.

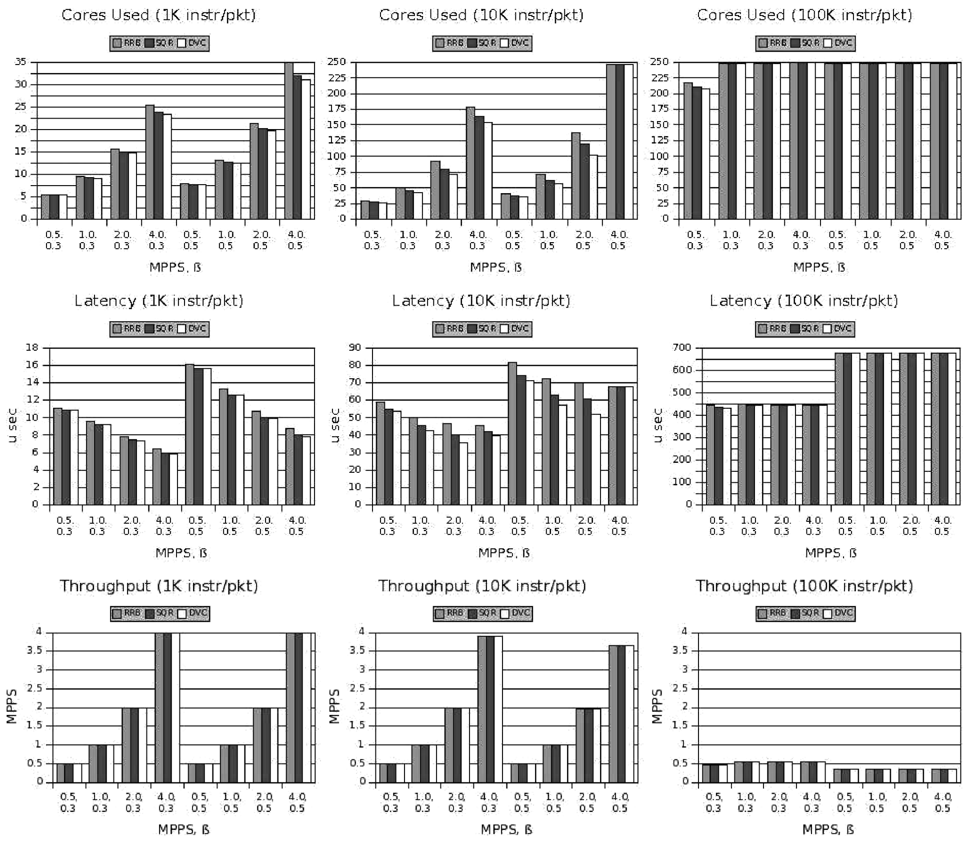


Fig. 8. Results for RRB, SQR and DVC as a function of $I(r)$, β, PKR and δ. PBL is fixed at 1 and CRF$_{max}$ is set to 10. The rest of parameters have the values listed in Table I.

served by remote cores, and thus the fewer, more expensive, accesses will be served by external memory. As the ingress date increases, so does the number of average cores used, implying that more packets of the same flow are currently processed.

—DVC always presents lower latency than both RRB and SQR. SQR has the potential for low latency since cores are selected so they are always close to each other. The latency reduction percentage of DVC versus SQR (RRB) is usually in the single digit range, and up to 14% (25%) for the ($I(r) = 10$K, $\beta = 0.5$, $\delta = 0.6$, PKR $= 2$, PBL $= 1$, CRF$_{max} = 10$) case.

—As the number of cores utilized reaches saturation (approaching 256), the benefit of DVC disappears since the probability of scheduling a packet to the first-choice virtual cluster diminishes because it is found full (no available core) most of the time.

—the overall throughput is similar for the three techniques. For those scenarios with low core utilization, more throughput is traded off for more active cores. For the high core utilization scenario (mainly the $I(r) = 100$K cases), the

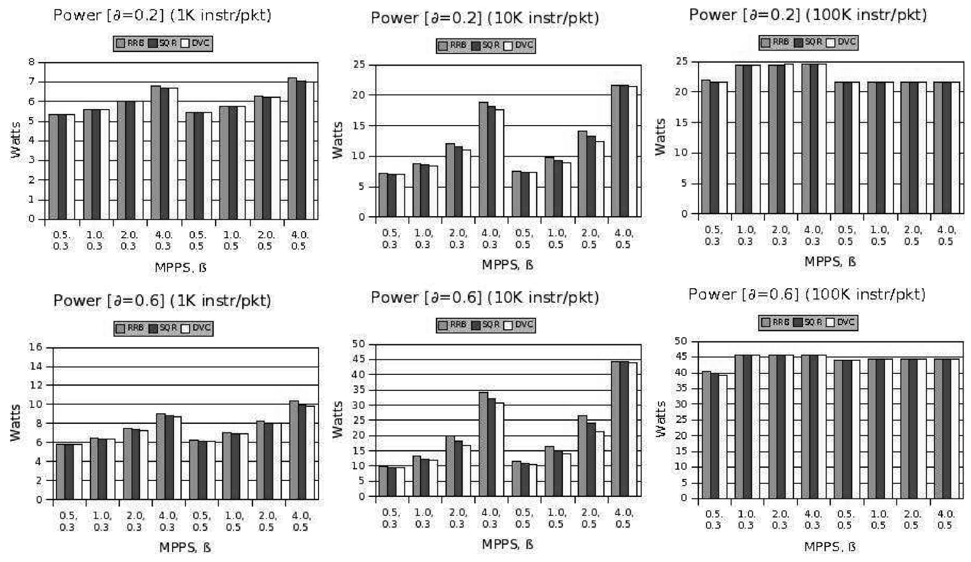


Fig. 9. Results for RRB, SQR and DVC as a function of $I(r)$, β, PKR and δ. PBL is fixed at 1 and CRF_{max} is set to 10. The rest of parameters have the values listed in Table I.

diminishing benefits of DVC renders all techniques having similar throughput characteristics. Throughput saturates at about 0.55 MPPS and 0.35 MPPS for $\beta = 0.3$ and $\beta = 0.5$ respectively for PKR ≥ 1 MPPS.

Regarding Power.

—The lower latency of DVC translates into a lower overall power consumption since fewer cores are active and therefore there are more opportunities for power gating.

—However, the amount of power savings is not as high as the latency reduction achieved. The reason is twofold: (a) a core that is waiting for remote data is clock gated so it consumes only static power, and (b) there is a fixed (I/O blocks) and variable (routers) power overhead, so the core-only power accounts for only a fraction of the overall power. Thus, the power savings of DVC vs SQR (RRB) for the same case singled out before is 10% (19%).

—Because of the clock gating performed by cores waiting for data and the routers being powered on all the time, the overall power consumption is significantly larger for a δ value of 0.6 versus 0.2.

Figure 10 analyzes in more detail one set of cases ($I(r) = 10\text{K}$, $\beta = 0.5$, $\delta = 0.6$, PKR $= 2$) when PBL and CRF_{max} are varied (Figures 8 and 9) showed the case for PBL $= 1$ and $\text{CRF}_{max} = 10$). Here PBL takes values 1, 4, and 16, and CRF_{max} takes 10, 50 and 100. Moreover, we just compare DVC against SQR since it has been shown that SQR is better than RRB in terms of power and performance. The plots show the absolute values for power and latency, and the percentage reduction by DVC. We can extrapolate that for low CRF_{max} values,

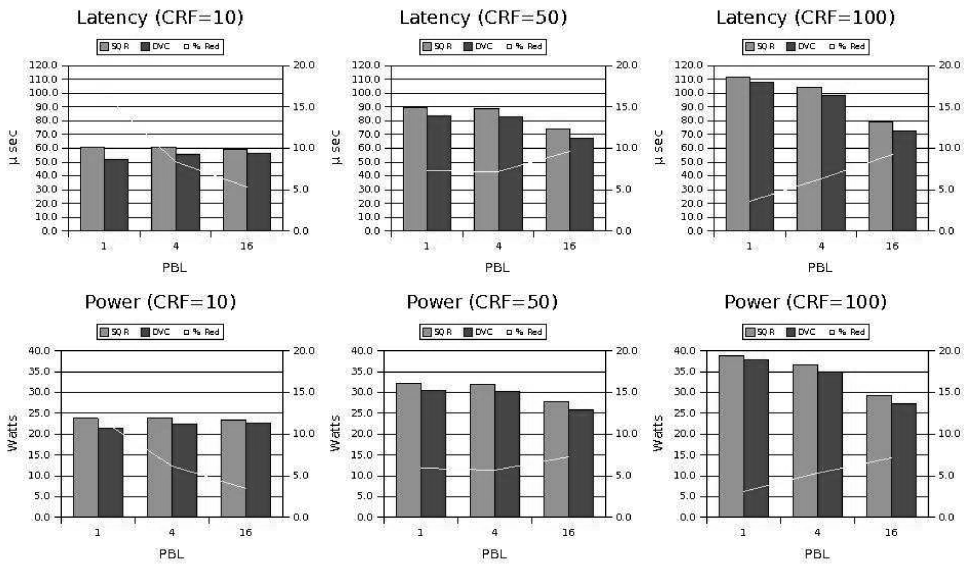


Fig. 10. Results for SQR and DVC for the ($I(r)$ = 10K, $\beta = 0.5$, $\delta = 0.6$, PKR = 2) case, varying PBL and CRF$_{max}$.

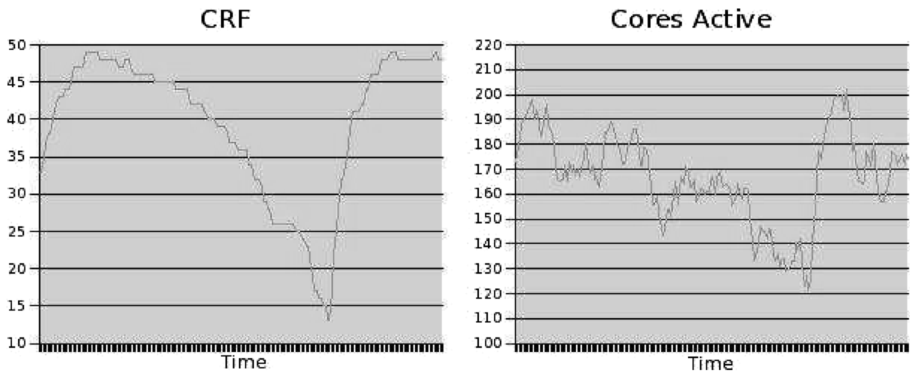
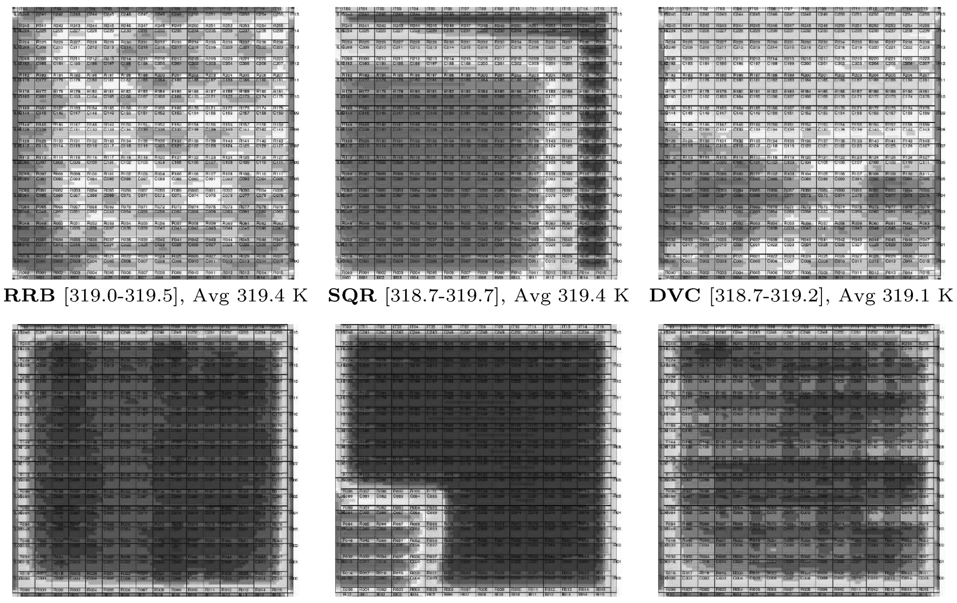


Fig. 11. Variation of CRF and cores used for a short period of time (2 ms) for the ($I(r)$ = 10K, $\beta = 0.5$, $\delta = 0.6$, PKR = 2, PBL = 4, CRF$_{max}$ = 50) case.

the benefits of DVC diminish as PBL increases, but the trend is the opposite for large CRF$_{max}$ values. This behavior is expected since DVC is most efficient in cases with large number of currently running flows, where packets can be segregated into many virtual clusters. For small CRF$_{max}$, SQR becomes efficient especially when PBL is large. In all cases except for PBL = 1 and CRF$_{max}$ = 10, the percentage reduction in both latency and power is in the single digit range.

Figure 11 plots the runtime CRF and cores used for a short period of time (2 ms) for the ($I(r)$ = 10K, $\beta = 0.5$, $\delta = 0.6$, PKR = 2, PBL = 4, CRF$_{max}$ = 50) case.[7] Core utilization presents greater variation than CRF since the number of

[7]The δ parameter does not affect this experiment.



RRB [319.0-319.5], Avg 319.4 K **SQR** [318.7-319.7], Avg 319.4 K **DVC** [318.7-319.2], Avg 319.1 K

RRB [314.9-315.5], Avg 315.0 K **SQR** [314.8-315.8], Avg 315.0 K **DVC** [314.9-315.5], Avg 315.0 K

Fig. 12. Thermal plots of the 256-core die (and I/O blocks) for the ($I(r)$ = 10K, β = 0.5, δ = 0.6, PKR = 2, PBL = 4, CRF$_{max}$ = 50) case (left column) and the ($I(r)$ =1K, β = 0.5, δ = 0.2, PKR = 4, PBL = 16, CRF$_{max}$ = 100) case (right column). Color coding is Red = Hot, Yellow = Warm, Green = Cool, Blue = Cold (range applies within each plot, not across plots, even though a rough comparison of all the plots based on color can be made). The min, max and average temperature is shown below each plot.

packets being concurrently processed across all currently running flows varies over time due to the burstiness of the incoming traffic. Nevertheless, as expected, the number of active cores broadly tracks the CRF runtime value.

Finally, Figure 12 shows the thermal plots for ($I(r)$ = 10K, β = 0.5, δ = 0.6, PKR = 2, PBL = 4, CRF$_{max}$ = 50) and the lower power case ($I(r)$ = 1 K, β = 0.5, δ = 0.2, PKR = 4, PBL = 16, CRF$_{max}$ = 100). We can observe that, as expected, SQR presents the worst thermal profile with a large hotspot on the lower left corner. On the other hand, RRQ presents the smoothest temperature distribution, while DVC features a thermal behavior very close to the optimal RRB. Note that the important result is not in the reduction, if any, of the average temperature, but rather in the reduction of (large) hotspots that can be observed in the ther mal plots. All these thermal simulations have been obtained with the HotSpot v3.0 tool [Skadron et al. 2003] using the default configuration parameters.

5. PREVIOUS WORK

Multi-core architectures and their power profile has been investigated extensively in the past years [Monchiero et al. 2006; Li et al. 2006; Merkel et al. 2005; Kursun et al. 2004; Choi et al. 2007; Chakraborty et al. 2007]. Power consumption in tile-based multi-core architecture has been addressed by a few works [Eisley et al. 2006; Oliver et al. 2006; Huang et al. 2008].

Migration of the processing activity has been proposed in several works [Heo et al. 2003; Chaparro et al. 2004; Donald and Martonosi 2006; M. Gomaa and Vijaykumar 2004; Huang et al. 2008] in order to spread the heat more uniformly across the die. The main drawback with this technique is the significant overhead in area (since the units have to be replicated in Heo et al. [2003]) and/or performance (since the state of the c omputation has to be explicitly copied to the new processing unit. Moreover, in Donald and Martonosi [2006] the complexity of the migration decision algorithm is proportional to the numb er of cores). Our work does not have these overheads because it follows a simple, yet effective in terms of throughput, run-to-completion service model where processing is not migrated.

Pande et al. [2005] is a comparison of different interconnect fabrics that can be implemented in an SoC, and in particular the mesh fabric that we use in our work. The paper validates our claim that a mesh-based interconnect provides high throughput at a low energy cost, and that at high degree of traffic localization (where the communication is mainly local to close neighbors) the energy per event (packet in this case) decreases. However, the paper does not study how this localization degree affects the temperature of the chip.

The specific effect that load balancing incoming requests has on the power and thermal characteristics in these tile-based architectures has been studied in Musoll [2008] and Stavrou and Trancoso [2007]. Flow-based load balancing in multi-core architectures has been addressed in Shi et al. [2005].

In both Musoll [2008] and Stavrou and Trancoso [2007] the authors assume totally independent tasks (i.e., no remote accesses are modeled, which affect the request runtime and therefore its power consumption). Moreover, the techniques presented are flow-agnostic (in other words, all packets or requests are assumed to belong to a single huge flow). Stavrou and Trancoso [2007] do not take also into account the performance impact of core communication nor models the layout and power of the router blocks. Moreover, the authors use the HotSpot tool to drive the thermal-aware load-balancing algorithm, which incurs a large performance overhead especially under lightweight workloads. In our work, we do not need any temperature estimation, just the simple task of keeping track of the number of currently running flows. Note that a precise temperature estimation requires on-chip sensors that require a large overhead since validation of the sensor outputs may need advance pattern recognition techniques to compensate their inaccuracy, or use more accurate but bulkier sensors, limiting the number of on-chip sensors that can be inserted [Lee et al. 2005; Coskun et al. 2007]. In a tile-based multi-core architecture, ideally there should be one sensor per core, which is very costly if not impossible with a massive number of cores (the maximum number of sensors is currently 27 in IBM's POWER5 processor). Gunther et al. [2001] use profiling to determine the best location for a single on-chip sensor. However, in massive multi-core processors, the location of the hotspot(s) is less deterministic, and it highly depends on the load balancing of the workload.

Shi et al. [2005] evaluate a system with up to 32 cores, leveraging the burst-like traffic to assign bursts of packets of a given flow to a core, while different bursts of the same flow can be scheduled to different cores to reduce the workload imbalance. In our work we take advantage of the large number of cores to assign packets to cores at line rate, thus decreasing buffering space and control logic complexity, rather than assigning a set of packets to the same core. Moreover, we provide estimations on the power and thermal behavior of our load-balancing technique and other simpler ones.

6. CONCLUSIONS

Future tile-based, massive multi-core architectures processing flow-based packet workloads will feature hundreds of cores to exploit the large amount of thread-level parallelism across flows. Packets within a flow, however, present dependencies that diminish their efficient processing in a multi-core architecture. This work presents a high-level study of the performance, power and thermal behavior of tile-based architectures with a large number of cores executing flow-based packet workloads, and proposes a load-balancing policy of assigning packets to cores that minimizes the communication latency while featuring a hotspot-free thermal profile.

REFERENCES

AGARWAL, A., BAO, L., BROWN, J., EDWARDS, B., MATTINA, M., MIAO, C., RAMEY, C., AND WENTZLAFF, D. 2007. Tile processor: Embedded multicore for networking and multimedia. In *Proceedings of the Symposium on High Performance Chips (Hot Chips)*.

AGARWAL, A., MUKHOPADHYAY, S., RAYCHOWDHURY, A., ROY, K., AND KIM, C. 2006. Leakage power analysis and reduction for nanoscale circuits. *IEEE Micro 26*, 2.

AZIMI, M., CHERUKURI, N., JAYASIMHA, D., KUMAR, A., KUNDU, P., PARK, S., SCHOINAS, I., AND VAIDYA, A. 2007. Integration challenges and tradeoffs for tera-scale architectures. *Intel. Tech. J. 11*, 3.

BALFOUR, J. AND DALLY, W. 2006. Design tradeoffs for tiles CMP networks. In *Proceedings of the International Conference on Supercomputing*.

BORKAR, S., JOUPPI, N., AND STENSTROM, P. 2007. Microprocessors in the era of terascale integration. In *Proceedings of the Conference and Exhibition on Design, Automation, and Test in Europe*.

CHAKRABORTY, K., WELLS, P., AND SOHI, G. 2007. A case for an over-provisioned multicore system: energy efficient processing of multithreaded programs. Tech. Rep. CS-TR-2007-1607. University of Wisconsin.

CHAPARRO, P., GONZALEZ, J., AND GONZALEZ, A. 2004. Thermal-aware clustered microarchitectures. In *Proceedings of the International Conference on Computer Design*.

CHOI, J., CHER, C.-Y., AND FRANKE, H. 2007. Thermal-aware task scheduling at the system software level. In *Proceedings of the International Symposium on Low Power Electronics and Design*.

COSKUN, A., ROSING, T., AND WHISNANT, K. 2007. Temperature aware task scheduling for MPSoCs. In *Proceedings of the Conference and Exhibition on Design, Automation, and Test in Europe*.

DALLY, W. AND TOWLES, B. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann.

DONALD, J. AND MARTONOSI, M. 2006. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the International Symposium on Computer Architecture*.

EISLEY, N., SOTERIOU, V., AND PEH, L.-S. 2006. High-level power analysis for multi-core chips. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*.

GOMAA, M., POWELL, M. D., AND VIJAYKUMAR, T. 2004. Heat-and-run: leveraging SMT and CMP to manage power density through the operating system. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*.

GUNTHER, S., BINNS, F., CARMEAN, D. M., AND HALL, J. C. 2001. Managing the impact of increasing microprocessor power consumption. *Intel Tech. J. 5*, 1.

HEO, S., BARR, K., AND ASANOVIC, K. 2003. Reducing power density through activity migration. In *Proceedings of the International Symposium on Low Power Electronics and Design*.

HOSKOTE, Y., VANGAL, S., SINGH, A., BORKAR, N., AND BORKAR, S. 2007. 5 GHz mesh interconnect for a teraflops processor. *IEEE Micro 27*, 5.

HU, Z., BUYUKTOSUNOGLU, A., SRINIVASAN, V., ZYUBAN, V., JACOBSON, H., AND BOSE, P. 2004. Microarchitectural techniques for power gating of executing units. In *Proceedings of the International Symposium on Low Power Electronics and Design*.

HUANG, W., STAN, M., SANKARANARAYANAN, K., RIBANDO, R., AND SKADRON, K. 2008. Many-core design from a thermal perspective. In *Proceedings of the Design Automation Conference*.

JAIN, R. 1992. A comparison of hashing schemes for address lookup in computer networks. *IEEE Trans. Comm. 40*, 3.

JIANG, H. AND DOVROLIS, C. 2003. Source-level IP packet bursts: causes and effects. In *Proceedings of the Conference on Internet Measurement*.

KEATING, M., FLYNN, D., AITKEN, R., GIBBONS, A., AND SHI, K. 2007. *Low Power Methodology Manual: For System-on-Chip Design*. Springer.

KEROMYTIS, A. AND PREVELAKIS, V. 2007. Designing firewalls: A survey. In *Network Security: Current Status and Future Directions*. IEEE Press.

KIM, J., BALFOUR, J., AND DALLY, W. 2007. Flattened butterfly topology for on-chip networks. In *Proceedings of the International Symposium on Microarchitecture*.

KIM, N., AUSTIN, T., BLAAUW, D., MUDGE, T., FLAUTNER, K., HU, J., IRWIN, M., KANDEMIR, M., AND NARAYANAN, V. 2003. Leakage current: Moore's Law meets static power. *IEEE Comput*.

KONSTANTAKOPOULOS, T., EASTEP, J., PSOTA, J., AND ARGAWAL, A. 2007. Energy scalability of on-chip interconnection networks in multicore architectures. Tech. rep., MIT CSAIL.

KURSUN, E., REINMAN, G., SAIR, S., SHAYESTEH, A., AND SHERWOOD, T. 2004. Low-overhead core swapping for thermal management. In *Proceedings of the Workshop on Thermal-Aware Computer Systems*.

LEE, K.-J., SKADRON, K., AND HUANG, W. 2005. Analytical model for sensor placement on microprocessors. In *Proceedings of the International Conference on Computer Design*.

LI, Y., LI, C., BROOKS, D., HU, Z., AND SKADRON, K. 2006. CMP design space exploration subject to physical constraints. In *Proceedings of the International Symposium on High Performance Computer Architecture*.

MERKEL, A., BELLOSA, F., AND WEISSEL, A. 2005. Event-driven thermal management in SMP systems. In *Proceedings of the Workshop on Thermal-Aware Computer Systems*.

MONCHIERO, M., CANAL, R., AND GONZALEZ, A. 2006. Design space exploration for multicore architectures: A power/performance/thermal view. In *Proceedings of the International Conference on Supercomputing*.

MULLINS, R., WEST, A., AND MOORE, S. 2004. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the International Symposium on Computer Architecture*.

MURALIMANOHAR, N. AND BALASUBRAMONIAN, R. 2007. Interconnect design considerations for large NUCA caches. In *Proceedings of the International Symposium on Computer Architecture*.

MUSOLL, E. 2008. A thermal-friendly load-balancing technique for multi-core processors. In *Proceedings of the International Symposium on Quality Electronic Design*.

OLIVER, J., RAO, R., BROWN, M., MANKIN, J., FRANKLIN, D., CHONG, F., AND AKELLA, V. 2006. The size selection for low-power tile-based architectures. In *Proceedings of the Conference on Computing Frontiers*.

PANDE, P. P., GRECU, C., JONES, M., IVANOV, A., AND SALEH, R. 2005. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Trans. Comput. 54*, 8.

PEH, L.-S. AND DALLY, W. 2001. A delay model and speculative architecture for pipelined routers. In *Proceedings of the International Symposium on High Performance Computer Architecture*.

ROY, K. 1998. Leakage power reduction in low-voltage CMOS design. In *Proceedings of the International Conference on Electronics, Circuits and Systems*.

SHI, W., MACGREGOR, M. H., AND GBURZYNSKI, P. 2003. Effects of a hash-based scheduler on cache performance in a parallel forwarding system. In *Proceedings of the Conference on Communication Networks and Distributed Systems Modeling and Simulation*.

SHI, W., MACGREGOR, M. H., AND GBURZYNSKI, P. 2005. A scalable load balancer for forwarding internet traffic: exploiting flow-level burstiness. In *Processings on the Symposium on Architectures for Networking and Communication Systems*.

SKADRON, K., STAN, M., HUANG, W., AND VELUSAMY, S. 2003. Temperature-aware microarchitecture. In *Proceedings of the International Symposium on Computer Architecture*.

STAVROU, K. AND TRANCOSO, P. 2007. Thermal-aware scheduling for future chip multiprocessors. *EURASIP J. Embed. Syst.*

VANGAL, S., HOWARD, J., RUHL, G., DIGHE, S., WILSON, H., TSCHANZ, J., FINAN, D., IYER, P., Y.HOSKOTE, AND BORKAR, N. 2007. An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS. In *Proceedings of the International Symposium on Solid State Circuit*.

Received August 2008; revised January 2009, May 2009; accepted May 2009